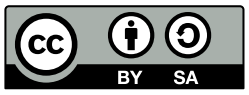
The Leading Open Source Backup Solution

Bacula® Main Reference Manual

Kern Sibbald

May 2, 2023

This manual documents Bacula Community Edition 13.0.3 (02 May 2023)



Bacula® is a registered trademark of Kern Sibbald

2023-05-02

version 13.0.3

Bacula Community

Contents

List of Figures

List of Tables

Chapter 1

What is Bacula Community Edition?

Bacula is a set of computer programs that permits the system administrator to manage backup, recovery, and verification of computer data across a network of computers of different kinds. Bacula can also run entirely upon a single computer and can backup to various types of media, including tape and disk.

In technical terms, it is a network Client/Server based backup program. Bacula is relatively easy to use and efficient, while offering many advanced storage management features that make it easy to find and recover lost or damaged files. Due to its modular design, Bacula is scalable from small single computer systems to systems consisting of hundreds of computers located over a large network.

1.1 Who Needs Bacula?

If you are currently using a program such as `tar`, `dump`, or `bru` to backup your computer data, and you would like a network solution, more flexibility, or catalog services, Bacula will most likely provide the additional features you want. However, if you are new to Unix systems or do not have offsetting experience with a sophisticated backup package, the Bacula project does not recommend using Bacula as it is much more difficult to setup and use than `tar` or `dump`.

If you want Bacula to behave like the above mentioned simple programs and write over any tape that you put in the drive, then you will find working with Bacula difficult. Bacula is designed to protect your data following the rules you specify, and this means reusing a tape only as the last resort. It is possible to "force" Bacula to write over any tape in the drive, but it is easier and more efficient to use a simpler program for that kind of operation.

If you would like a backup program that can write to multiple volumes (i.e. is not limited by your tape drive capacity), Bacula can most likely fill your needs. In addition, quite a number of Bacula users report that Bacula is simpler to setup and use than other equivalent programs.

If you are currently using a sophisticated commercial package such as Legato Networker. ARCserveIT, Arkeia, or PerfectBackup+, you may be interested in Bacula, which provides many of the same features and is free software available under the Affero GPL Version 3 software license.

1.2 Bacula Components or Services

Bacula is made up of the following five major components or services: Director, Console, File, Storage, and Monitor services.

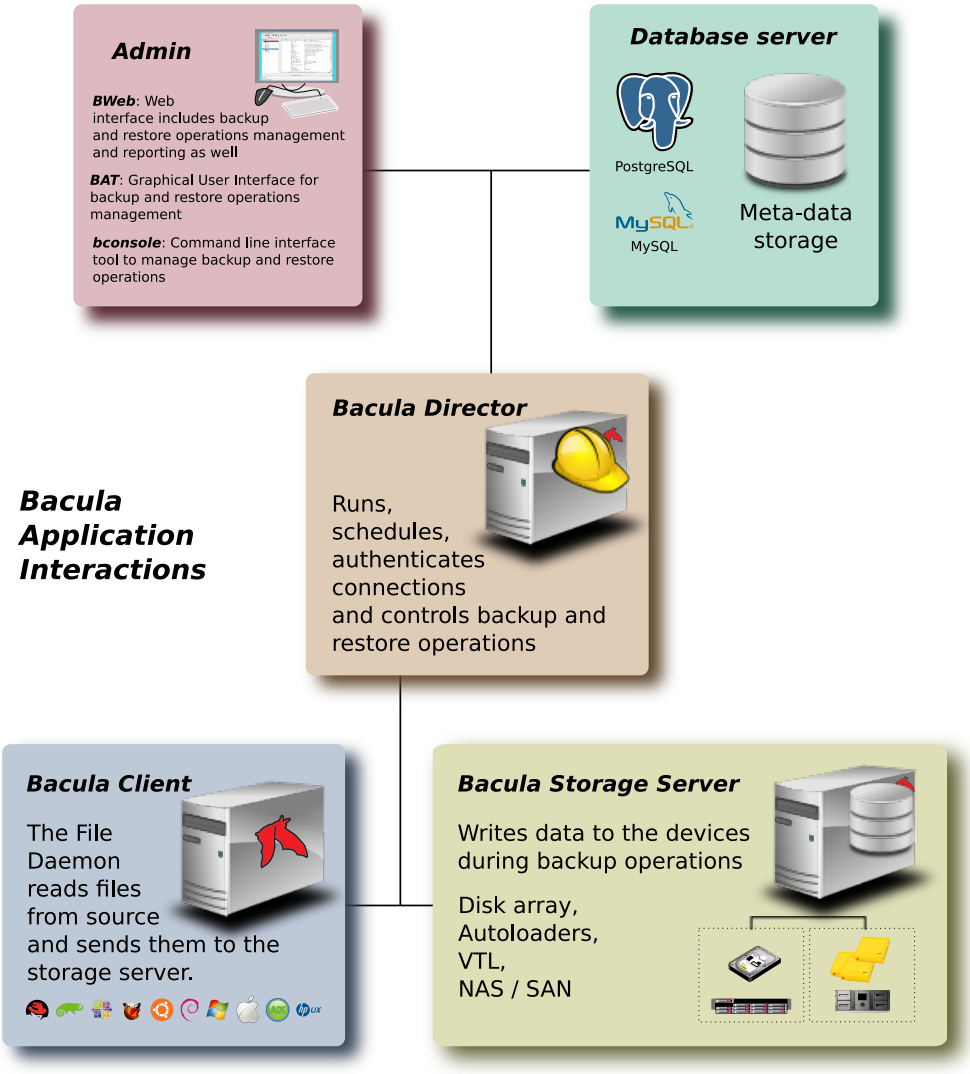


Figure 1.1: Bacula Applications

(thanks to Aristedes Maniatis for this graphic and the one below)

Bacula Director

The Bacula Director service is the program that supervises all the backup, restore, verify and archive operations. The system administrator uses the Bacula Director to schedule backups and to recover files. For more details see the **Director Services Daemon Design** (chapter 6 page 59) in the Bacula Community Edition Developer's manual. The Director runs as a daemon (or service) in the background.

Bacula Console

The Bacula Console service is the program that allows the administrator or user to communicate with the Bacula Director Currently, the Bacula Console is available in three versions: text-based console interface, QT-based interface, and a wxWidgets graphical interface. The first and simplest is to run the Console program in a shell window (i.e. TTY interface). Most system administrators will find this completely adequate. The second version is a GNOME GUI interface that is far from complete, but quite functional as it has most the capabilities of the shell Console. The third version is a wxWidgets GUI with an interactive file restore. It also has most of the capabilities of the shell console, allows command completion with tabulation, and gives you instant help about the command you are typing. For more details see the **Bacula Console Design Document** (Chapter 1 page 1).

Bacula File

The Bacula File service (also known as the Client program) is the software program that is installed on the machine to be backed up. It is specific to the operating system on which it runs and is responsible for providing the file attributes and data when requested by the Director. The File services are also responsible for the file system dependent part of restoring the file attributes and data during a recovery operation. For more details see the **File Services Daemon Design** (chapter 7 page 61) in the Bacula Community Edition Developer's manual. This program runs as a daemon on the machine to be backed up. In addition to Unix/Linux File daemons, there is a Windows File daemon (normally distributed in binary format). The Windows File daemon runs on current Windows versions (NT, 2000, XP, 2003, and possibly Me and 98).

Bacula Storage

The Bacula Storage services consist of the software programs that perform the storage and recovery of the file attributes and data to the physical backup media or volumes. In other words, the Storage daemon is responsible for reading and writing your tapes (or other storage media, e.g. files). For more details see the **Storage Services Daemon** (chapter 8 page 63) in the Bacula Community Edition Developer's manual. The Storage services runs as a daemon on the machine that has the backup device (usually a tape drive).

Catalog

The Catalog services are comprised of the software programs responsible for maintaining the file indexes and volume databases for all files backed up. The Catalog services permit the system administrator or user to quickly locate and restore any desired file. The Catalog services sets Bacula apart from simple backup programs like `tar` and `bru`, because the catalog maintains a record of all Volumes used, all Jobs run, and all Files saved, permitting efficient restoration and Volume management. Bacula currently supports three different databases, `MySQL`, `PostgreSQL`, one of which must be chosen when building Bacula.

The three SQL databases currently supported (`MySQL`, `PostgreSQL`) quite a number of features, including rapid indexing, arbitrary queries, and security. Although the Bacula project plans to support other major SQL databases, the current Bacula implementation interfaces only to `MySQL`, `PostgreSQL`. For the technical and porting details see the **Catalog Services Design** (chapter 9 page 67) in the Bacula Community Edition Developer's manual.

The packages for `MySQL` and `PostgreSQL` are available for several operating systems. Alternatively, installing from the source is quite easy, see the Installing and Configuring `MySQL` chapter of this document for the details. For more information on `MySQL`, please see: www.mysql.com. Or see the Installing and Configuring `PostgreSQL` chapter of this document for the details. For more information on `PostgreSQL`, please see: www.postgresql.org.

Bacula Monitor

A Bacula Monitor service is the program that allows the administrator or user to watch current status of Bacula Directors, Bacula File Daemons and Bacula Storage Daemons. Currently, only a GTK+ version is available, which works with GNOME, KDE, or any window manager that supports the FreeDesktop.org system tray standard.

To perform a successful save or restore, the following four daemons must be configured and running: the Director daemon, the File daemon, the Storage daemon, and the Catalog service (`MySQL` or `PostgreSQL`).

1.3 Bacula Configuration

In order for Bacula to understand your system, what clients you want backed up and how, you must create a number of configuration files containing resources (or objects). The following presents an overall picture of this:

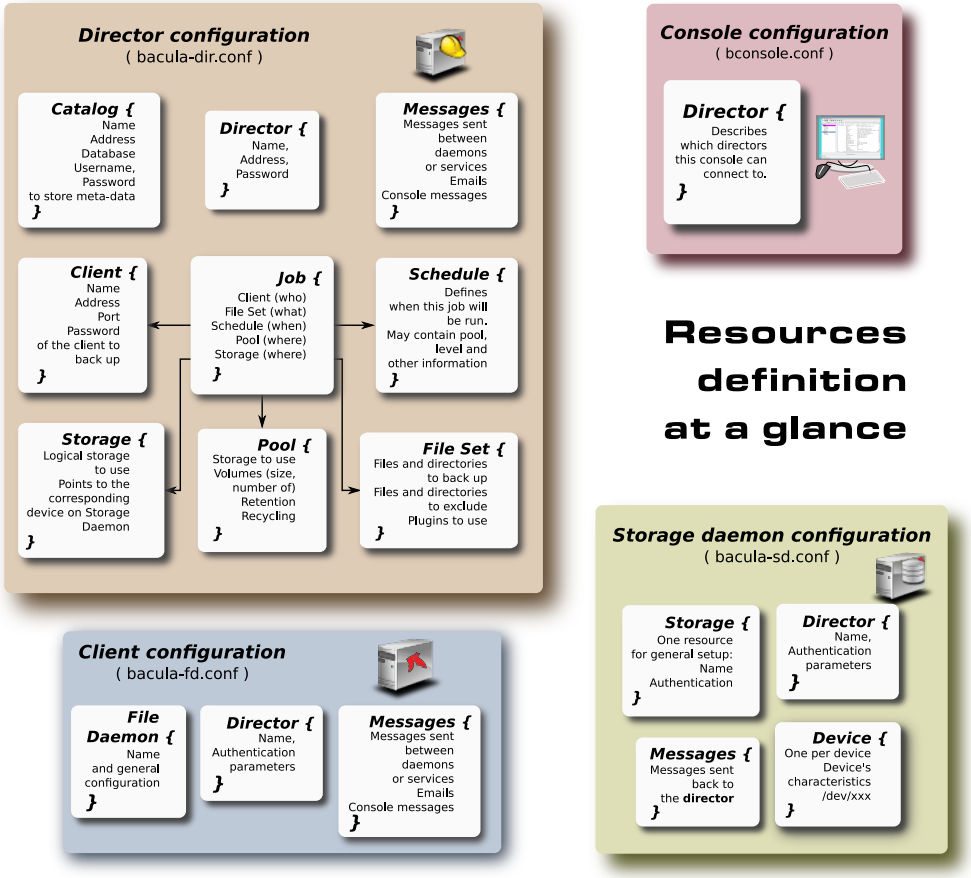


Figure 1.2: Bacula Objects

1.4 Conventions Used in this Document

Bacula is in a state of evolution, and as a consequence, this manual will not always agree with the code. If an item in this manual is preceded by an asterisk (*), it indicates that the particular

feature is not implemented. If it is preceded by a plus sign (+), it indicates that the feature may be partially implemented.

If you are reading this manual as supplied in a released version of the software, the above paragraph holds true. If you are reading the online version of the manual, www.bacula.org, please bear in mind that this version describes the current version in development (in the git repository) that may contain features not in the released version. Just the same, it generally lags behind the code a bit.

1.5 Quick Start

To get Bacula up and running quickly, the author recommends that you first scan the Terminology section below, then quickly review the next chapter entitled The Current State of Bacula, then the Getting Started with Bacula, which will give you a quick overview of getting Bacula running. After which, you should proceed to the chapter on Installing Bacula, then How to Configure Bacula, and finally the chapter on Running Bacula.

1.6 Terminology

Administrator The person or persons responsible for administrating the Bacula system.

Backup The term Backup refers to a Bacula Job that saves files.

Bootstrap File The bootstrap file is an ASCII file containing a compact form of commands that allow Bacula or the stand-alone file extraction utility (`bextract`) to restore the contents of one or more Volumes, for example, the current state of a system just backed up. With a bootstrap file, Bacula can restore your system without a Catalog. You can create a bootstrap file from a Catalog to extract any file or files you wish.

Catalog The Catalog is used to store summary information about the Jobs, Clients, and Files that were backed up and on what Volume or Volumes. The information saved in the Catalog permits the administrator or user to determine what jobs were run, their status as well as the important characteristics of each file that was backed up, and most importantly, it permits you to choose what files to restore. The Catalog is an online resource, but does not contain the data for the files backed up. Most of the information stored in the catalog is also stored on the backup volumes (i.e. tapes). Of course, the tapes will also have a copy of the file data in addition to the File Attributes (see below).

The catalog feature is one part of Bacula that distinguishes it from simple backup and archive programs such as `dump` and `tar`.

Client In Bacula's terminology, the word Client refers to the machine being backed up, and it is synonymous with the File services or File daemon, and quite often, it is referred to it as the FD. A Client is defined in a configuration file resource.

Console The program that interfaces to the Director allowing the user or system administrator to control Bacula.

Daemon Unix terminology for a program that is always present in the background to carry out a designated task. On Windows systems, as well as some Unix systems, daemons are called Services.

Directive The term directive is used to refer to a statement or a record within a Resource in a configuration file that defines one specific setting. For example, the **Name** directive defines the name of the Resource.

Director The main Bacula server daemon that schedules and directs all Bacula operations. Occasionally, the project refers to the Director as DIR.

Differential A backup that includes all files changed since the last Full save started. Note, other backup programs may define this differently.

File Attributes The File Attributes are all the information necessary about a file to identify it and all its properties such as size, creation date, modification date, permissions, etc. Normally, the attributes are handled entirely by Bacula so that the user never needs to be concerned about them. The attributes do not include the file's data.

File Daemon The daemon running on the client computer to be backed up. This is also referred to as the File services, and sometimes as the Client services or the FD.

FileSet A FileSet is a Resource contained in a configuration file that defines the files to be backed up. It consists of a list of included files or directories, a list of excluded files, and how the file is to be stored (compression, encryption, signatures). For more details, see the FileSet Resource definition in the Director chapter of this document.

Incremental A backup that includes all files changed since the last Full, Differential, or Incremental backup started. It is normally specified on the **Level** directive within the Job resource definition, or in a `Schedule` resource.

Job A Bacula Job is a configuration resource that defines the work that Bacula must perform to backup or restore a particular Client. It consists of the **Type** (backup, restore, verify, etc), the **Level** (full, incremental,...), the **FileSet**, and **Storage** the files are to be backed up (Storage device, Media Pool). For more details, see the Job Resource definition in the Director chapter of this document.

Monitor The program that interfaces to all the daemons allowing the user or system administrator to monitor Bacula status.

Resource A resource is a part of a configuration file that defines a specific unit of information that is available to Bacula. It consists of several directives (individual configuration statements). For example, the `Job` resource defines all the properties of a specific Job: name, schedule, Volume pool, backup type, backup level, ...

Restore A restore is a configuration resource that describes the operation of recovering a file from backup media. It is the inverse of a save, except that in most cases, a restore will normally have a small set of files to restore, while normally a Save backs up all the files on the system. Of course, after a disk crash, Bacula can be called upon to do a full Restore of all files that were on the system.

Schedule A Schedule is a configuration resource that defines when the Bacula Job will be scheduled for execution. To use the Schedule, the Job resource will refer to the name of the Schedule. For more details, see the Schedule Resource definition in the Director chapter of this document.

Service This is a program that remains permanently in memory awaiting instructions. In Unix environments, services are also known as **daemons**.

Storage Coordinates The information returned from the Storage Services that uniquely locates a file on a backup medium. It consists of two parts: one part pertains to each file saved, and the other part pertains to the whole Job. Normally, this information is saved in the Catalog so that the user doesn't need specific knowledge of the Storage Coordinates. The Storage Coordinates include the File Attributes (see above) plus the unique location of the information on the backup Volume.

Storage Daemon The Storage daemon, sometimes referred to as the SD, is the code that writes the attributes and data to a storage Volume (usually a tape or disk).

Session Normally refers to the internal conversation between the File daemon and the Storage daemon. The File daemon opens a **session** with the Storage daemon to save a FileSet or to restore it. A session has a one-to-one correspondence to a Bacula Job (see above).

Verify A verify is a job that compares the current file attributes to the attributes that have previously been stored in the Bacula Catalog. This feature can be used for detecting changes to critical system files similar to what a file integrity checker like Tripwire does. One of the major advantages of using Bacula to do this is that on the machine you want protected such as a server, you can run just the File daemon, and the Director, Storage daemon, and Catalog reside on a different machine. As a consequence, if your server is ever compromised, it is unlikely that your verification database will be tampered with.

Verify can also be used to check that the most recent Job data written to a Volume agrees with what is stored in the Catalog (i.e. it compares the file attributes), *or it can check the Volume contents against the original files on disk.

*Archive An Archive operation is done after a Save, and it consists of removing the Volumes on which data is saved from active use. These Volumes are marked as Archived, and may no longer be used to save files. All the files contained on an Archived Volume are removed from the Catalog. NOT YET IMPLEMENTED.

Retention Period There are various kinds of retention periods that Bacula recognizes. The most important are the **File** Retention Period, **Job** Retention Period, and the **Volume** Retention Period. Each of these retention periods applies to the time that specific records will be kept in the Catalog database. This should not be confused with the time that the data saved to a Volume is valid.

The File Retention Period determines the time that File records are kept in the catalog database. This period is important for two reasons: the first is that as long as File records remain in the database, you can "*browse*" the database with a console program and restore any individual file. Once the File records are removed or pruned from the database, the individual files of a backup job can no longer be "browsed". The second reason for carefully choosing the File Retention Period is because the volume of the database File records use the most storage space in the database. As a consequence, you must ensure that regular "*pruning*" of the database file records is done to keep your database from growing too large. (See the Console `prune` command for more details on this subject).

The Job Retention Period is the length of time that Job records will be kept in the database. Note, all the File records are tied to the Job that saved those files. The File records can be purged leaving the Job records. In this case, information will be available about the jobs that ran, but not the details of the files that were backed up. Normally, when a Job record is purged, all its File records will also be purged.

The Volume Retention Period is the minimum of time that a Volume will be kept before it is reused. Note, if all the Jobs and Files associated to a Volume are pruned from Catalog, Bacula may reuse this Volume before its retention time. Bacula will normally never overwrite a Volume that contains the only backup copy of a file. Under ideal conditions, the Catalog would retain entries for all files backed up for all current Volumes. Once a Volume is overwritten, the files that were backed up on that Volume are automatically removed from the Catalog. However, if there is a very large pool of Volumes or a Volume is never overwritten, the Catalog database may become enormous. To keep the Catalog to a manageable size, the backup information should be removed from the Catalog after the defined File Retention Period. Bacula provides the mechanisms for the catalog to be automatically pruned according to the retention periods defined.

Scan A Scan operation causes the contents of a Volume or a series of Volumes to be scanned. These Volumes with the information on which files they contain are restored to the Bacula Catalog. Once the information is restored to the Catalog, the files contained on those Volumes may be easily restored. This function is particularly useful if certain Volumes or Jobs have exceeded their retention period and have been pruned or purged from the Catalog. Scanning data from Volumes into the Catalog is done by using the `bscan` program. See the **bscan** section (section 1.7 page 7) of the Bacula Community Edition Utility programs for more details.

Volume A Volume is an archive unit, normally a tape or a named disk file where Bacula stores the data from one or more backup jobs. All Bacula Volumes have a software label written to the Volume by Bacula so that it identifies what Volume it is really reading. (Normally

there should be no confusion with disk files, but with tapes, it is easy to mount the wrong one.)

1.7 What Bacula is Not

Bacula is a backup, restore and verification program and is not a complete disaster recovery system in itself, but it can be a key part of one if you plan carefully and follow the instructions included in the Disaster Recovery Chapter of this manual.

With proper planning, as mentioned in the Disaster Recovery chapter, Bacula can be a central component of your disaster recovery system. For example, if you have created an emergency boot disk, and/or a Bacula Rescue disk to save the current partitioning information of your hard disk, and maintain a complete Bacula backup, it is possible to completely recover your system from "bare metal" that is starting from an empty disk.

If you have used the **WriteBootstrap** record in your job or some other means to save a valid bootstrap file, you will be able to use it to extract the necessary files (without using the catalog or manually searching for the files to restore).

1.8 Interactions Between the Bacula Services

The block diagram 1.3 on the facing page shows the typical interactions between the Bacula Services for a backup job. Each block represents in general a separate process (normally a daemon). In general, the Director oversees the flow of information. It also maintains the Catalog.

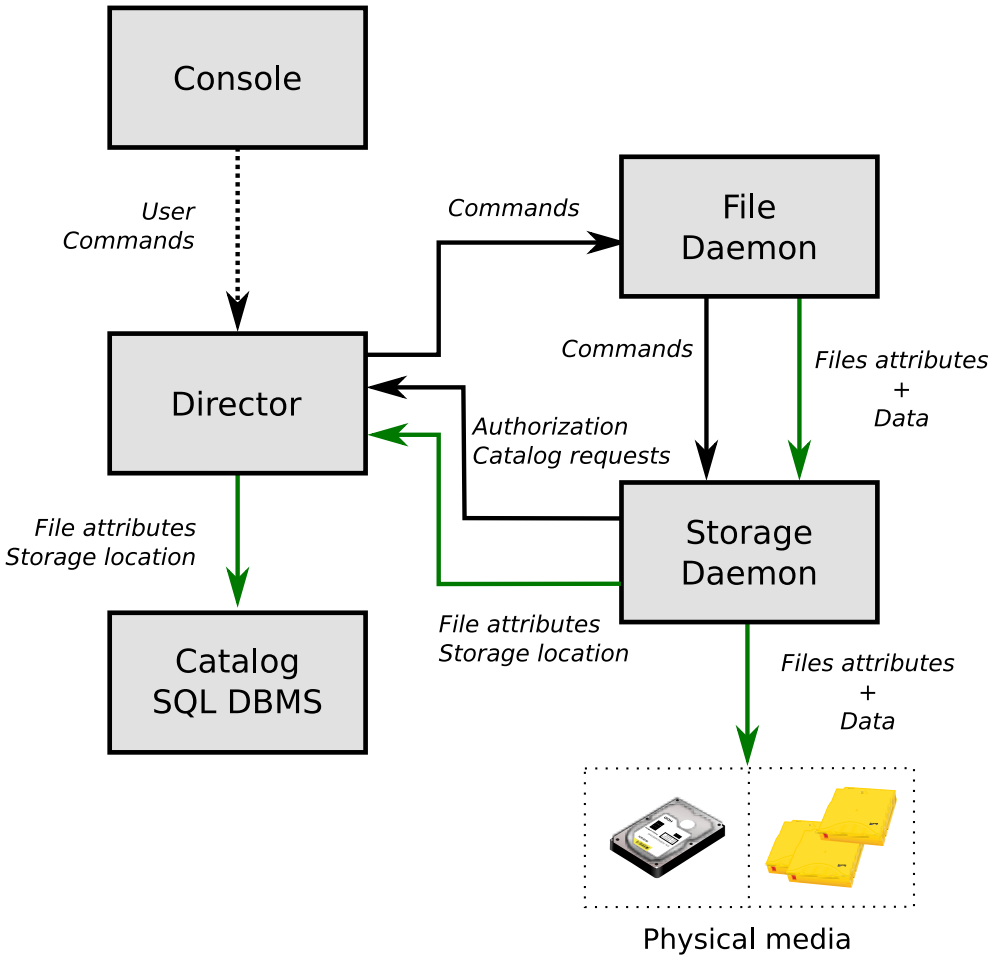


Figure 1.3: Interactions between Bacula Services

Chapter 2

New Features in 13.0.0

2.0.1 Kubernetes Plugin

The Bacula Kubernetes plugin will save all the important Kubernetes resources which make up the application or service. The plugin has the following features:

- Kubernetes cluster resources configuration backup
- Ability to restore single Kubernetes configuration resource
- Ability to restore Kubernetes resource configuration to local directory
- Kubernetes Persistent Volumes data backup and restore
- Ability to restore Kubernetes Persistent Volumes data to local directory
- Ability to use new Kubernetes CSI driver snapshot features to perform Persistent Volume data backup
- Ability to execute user defined commands on required Pod containers to prepare and clean data backup
- Configure Kubernetes workload backup requirements directly with Pod annotations

More information about the Kubernetes plugin can be found on 48.1 on page 515.

2.0.2 Storage Group

It is now possible to configure more than one Storage resource for each Job/Pool. Storage can be specified as a comma separated list of Storage resources to use.

Along with specifying a Storage list it is now possible to specify a Storage Group Policy which will be used for determining which Storage will be used for the Job. If no policy is specified, Bacula tries to use the first available Storage from the list.

If the first few storage daemons are unavailable due to network problems; broken or unreachable for some other reason, Bacula will use the first one from the list (sorted according to the policy used) which is network reachable and healthy.

Currently supported policies are:

ListedOrder - This is the default policy. It uses the first available storage from the list provided

LeastUsed - This policy scans all storage daemons from the list and chooses the one with the least number of Jobs currently running

Storage Groups may be used as follows (as a part of Job and Pool resources):

```
Job {
   ...
   Storage = File1, File2, File3
   ...
}

Pool {
   ...
   Storage = File4, File5, File6
   StorageGroupPolicy = LeastUsed
   ...
}
```

When a Job or Pool with a Storage Group is used, the user will observe some messages related to the choice of Storage such as:

```
messages
31-maj 19:23 VBox-dir JobId 1: Start Backup JobId 1, Job=StoreGroupJob.2021-05-31_19.23.36_03
31-maj 19:23 VBox-dir JobId 1: Possible storage choices: "File1, File2"
31-maj 19:23 VBox-dir JobId 1: Storage daemon "File1" didn't accept Device "FileChgr1-Dev1" because: 3924 Device "Fi
31-maj 19:23 VBox-dir JobId 1: Selected storage: File2, device: FileChgr2-Dev1, StorageGroupPolicy: "ListedOrder"
```

2.0.3 New Accurate Option to Save Only File's Metadata

The new 'o' Accurate directive option for a Fileset has been added to save only the metadata of a file when possible.

The new 'o' option should be used in conjunction with one of the signature checking options (1, 2, 3, or 5). When the 'o' option is specified, the signature is computed only for files that have one of the other accurate options specified triggering a backup of the file (for example an inode change, a permission change, etc...).

In cases where only the file's metadata has changed (ie: the signature is identical), only the file's attributes will be backed up. If the file's data has been changed (hence a different singature), the file will be backed up in the usual way (attributes as well as the file's contents will be saved on the volume).

For example:

```
Job {
  Name = JobTest
  JobDefs = DefaultJob
  FileSet = TestFS
  Accurate = yes
}

FileSet {
  Name = TestFS
  Options {
    Signature = MD5
    Accurate = pino5
  }
  File = /
}
```

The backup job will compare permission bits, inodes, number of links and, if any of it changes, it will also compute file's signature to verify if only the metadata must be backed up or if the full file must be saved.

2.0.4 External LDAP Console Authentication

The new Bacula Plugable Authentication Module (BPAM) API framework introduced in Bacula 13.0 comes with the first plugin which handles user authentication against any LDAP Directory Server (including OpenLDAP and Active Directory). To enable the build of the LDAP Console Authentication plugin, add the `ldap` options to your `./configure` command line as shown below:

```
./configure --with-ldap --enable-ldap-bpam [... other options]
```

```
# bconsole
*status director
...
 Plugin: ldap-dir.so
...
```

When the LDAP plugin is loaded you can configure a named console resource to use LDAP to authenticate users. BConsole will prompt for a User and Password and it will be verified by the Director. TLS PSK (activated by default) is recommended. To use this plugin, you have to specify the `PluginDirectory` Director resource directive, and add a new console resource directive `Authentication Plugin` as shown below:

```
Director {
  ...
  Plugin Directory = /opt/bacula/plugins
}

Console {
   Name = "ldapconsole"
   Password = "xxx"

   # New directive
   Authentication Plugin = "ldap:<parameters>"
   ...
}
```

where `parameters` are the space separated list of one or more plugin parameters:

url - LDAP Directory service location, i.e. "url=ldap://10.0.0.1/"

binddn - DN used to connect to LDAP Directory service to perform required `query`

bindpass - DN password used to connect to LDAP Directory service to perform required `query`

query - A query performed on the LDAP Directory serice to find user for authentication The `query` string is composed as `<basedn>/<filter>`. Where '`<basedn>` is a DN search starting point and `<filter>` is a standard LDAP search object filter which support dynamic string substitution: %u will be replaced by credential's username and %p by credential's password, i.e. `query=dc=bacula,dc=com/(cn=%u)`.

starttls - Instruct the BPAM LDAP Plugin to use the **StartTLS** extension if the LDAP Directory service supports it and fallback to no TLS if this extension is not available.

starttlsforce - Does the same as the 'starttls' setting does but reports error on fallback.

Working configuration examples:

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with OpenLDAP authentication example.

```
Console {
 Name = "bacula_ldap_console"
 Password = "xxx"

 # New directive (on a single line)
 Authentication Plugin = "ldap: url=ldap://ldapsrv/ binddn=cn=root,dc=bacula,dc=com
                          bindpass=secret query=dc=bacula,dc=com/(cn=%u) starttls"
   ...
}
```

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with Active Directory authentication example.

```
Console {
 Name = "bacula_ad_console"
 Password = "xxx"

 # New directive (on a single line)
 Authentication Plugin = "ldap: url=ldaps://ldapsrv/
       binddn=cn=bacula,ou=Users,dc=bacula,dc=com bindpass=secret
       query=dc=bacula,dc=com/(&(objectCategory=person)(objectClass=user)(sAMAccountName=%u))"
   ...
}
```

2.0.5 New Baculum features

The following features were introduced in version 11.0.6

New Advanced Schedule Settings

The schedule function has been reworked to support all possible schedule settings. Now it can be configured in five different ways: hourly, daily, weekly, monthly, and also custom for various uncommon settings.

New Copy and Migrate Job Wizards

Using the new wizards it is possible to easily create new copy and migrate jobs. The Wizard's steps are prepared and optimized to make the process as easy as possible.

New Director Page

This new page provides two major functions: The Director status and the Director configuration. The first one is for displaying in a graphical and textual way the Director status. The second one is for configuring all Director resources (Jobs, Clients, Pools, Storage ...etc.). From one place users can configure various Director resources.

Bulk Delete Volumes Action

This feature is available on the media page. Apart from prune and purge bulk actions, now users are also able to bulk delete volumes. It is possible by selecting volumes in the volume table and using this new delete volumes action.

Loading Pages Optimization

Pages have been optimized to faster loading. The entire Baculum Web interface works faster.

Directive Documentation

In all Bacula component configurations, for almost all directives, official Bacula documentation has been added. It is available directly within the web interface.

2.0.6 Tag Support

It is now possible to assign custom Tags to various catalog records in Bacula such as:

- Volume
- Client
- Job

```
*tag
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
Available Tag operations:
     1: Add
     2: Delete
     3: List
Select Tag operation (1-3): 1
Available Tag target:
     1: Client
     2: Job
     3: Volume
Select Tag target (1-3): 1
The defined Client resources are:
     1: 127.0.0.1-fd
     2: test1-fd
     3: test2-fd
     4: test-rst-fd
     5: test-bkp-fd
Select Client (File daemon) resource (1-5): 1
Enter the Tag value: test1
1000 Tag added

*tag add client=127.0.0.1-fd name=#important"
1000 Tag added

*tag list client
+--------------+----------+--------------+
| tag          | clientid | client       |
+--------------+----------+--------------+
| #tagviamenu3 |        1 | 127.0.0.1-fd |
| test1        |        1 | 127.0.0.1-fd |
| #tagviamenu2 |        1 | 127.0.0.1-fd |
| #tagviamenu1 |        1 | 127.0.0.1-fd |
| #important   |        1 | 127.0.0.1-fd |
+--------------+----------+--------------+

*tag list client name=#important
+----------+--------------+
| clientid | client       |
+----------+--------------+
|        1 | 127.0.0.1-fd |
+----------+--------------+
```

It is possible to assign Tags to a Job record with the new 'Tag' directive in a Job resource.

```
Job {
  Name = backup
  ...
  Tag = "#important", "#production"
}
```

```
*tag list job
+--------------+-------+-------------+
| tag          | jobid | job         |
+--------------+-------+-------------+
| #important   |     2 | backup      |
| #production  |     2 | backup      |
+--------------+-------+-------------+
```

The Tags are also accessible from various BWeb Management Suite screens. (See **??** on page ??).

2.0.7 Support for SHA256 and SHA512 in FileSet

The support for strong signature algorithms SHA256 and SHA512 has been added to Verify Jobs. It is now possible to check if data generated by a Job that uses an SHA256 or SHA512 signature is valid.

```
FileSet {
  Name = Linux-Etc
  Options {
    Signature = SHA512
    Verify = pins3
  }
  File = /etc
}
```

In the `FileSet` **Verify** option directive, the following code has been added:

3 for SHA512

2 for SHA256

2.0.8 Windows Installer Silent Mode Enhancement

The following command line options may be used to control the regular Bacula installer values in silent mode:

- -ConfigClientName
- -ConfigClientPort
- -ConfigClientPassword
- -ConfigClientMaxJobs
- -ConfigClientInstallService
- -ConfigClientStartService
- -ConfigStorageName
- -ConfigStoragePort
- -ConfigStorageMaxJobs
- -ConfigStoragePassword
- -ConfigStorageInstallService

- -ConfigStorageStartService
- -ConfigDirectorName
- -ConfigDirectorPort
- -ConfigDirectorMaxJobs
- -ConfigDirectorPassword
- -ConfigDirectorDB
- -ConfigDirectorInstallService
- -ConfigDirectorStartService
- -ConfigMonitorName
- -ConfigMonitorPassword

The following options control the components that will be installed:

- -ComponentFile
- -ComponentStorage
- -ComponentTextConsole
- -ComponentBatConsole
- -ComponentTrayMonitor
- -ComponentAllDrivesPlugin
- -ComponentWinBMRPlugin
- -ComponentCDPPlugin

Example

```
bacula-win64-13.0.0.exe /S -ComponentFile \
    -ConfigClientName foo -ConfigClientPassword bar
```

Will install only file deamon with bacula-fd.conf configured.

```
bacula-win64-13.0.0.exe /S -ComponentStorage \
   -ComponentFile-ConfigClientName foo \
   -ConfigClientPassword bar -ConfigStorageName foo2 \
   -ConfigStoragePassword bar2
```

Will install the Storage Deamon plus File Deamon with bacula-sd.conf and bacula-fd.conf configured.

2.0.9 New Message Identification Format

We are starting to add unique message indentifiers to each message (other than debug and the Job report) that Bacula prints. At the current time only two files in the Storage Daemon have these message identifiers and over time with subsequent releases we will modify all messages.

The message identifier will be kept unique for each message and once assigned to a message it will not change even if the text of the message changes. This means that the message identifier will be the same no matter what language the text is displayed in, and more importantly, it will allow us to make listing of the messages with in some cases, an additional explanation or

instructions on how to correct the problem. All this will take several years since it is a lot of work and requires some new programs that are not yet written to manage these message identifiers.

The format of the message identifier is:

```
[AAnnnn]
```

where A is an upper case character and nnnn is a four digit number, where the first character indicates the software component (daemon); the second letter indicates the severity, and the number is unique for a given component and severity.

For example:

```
[SF0001]
```

The first character representing the component at the current time one of the following:

```
S       Storage daemon
D       Director
F       File daemon
```

The second character representing the severity or level can be:

```
A       Abort
F       Fatal
E       Error
W       Warning
S       Security
I       Info
D       Debug
O       OK (i.e. operation completed normally)
```

So in the example above [SF0001] indicates it is a message id, because of the brackets and because it is at the beginning of the message, and that it was generated by the Storage Daemon as a fatal error. As mentioned above it will take some time to implement these message ids everywhere, and over time we may add more component letters and more severity levels as needed.

2.0.10 Misc

Plugin Object Status Support

The Object table has now a ObjectStatus field that can be used by plugins to report more precise information about the backup of an Object (generated by plugins).

Network Buffer Management

The new `SDPacketCheck` FileDaemon directive can be used to control the network flow in some specific use cases.

See 23.1 on page 304 for more information.

IBM Lintape Driver (BETA)

The new `Use Lintape` Storage Daemon directive has been added to support the Lintape Kernel driver.

See 24.3 on page 331 for more information.

Chapter 3

New Features in 11.0.0

3.1 Catalog Performance Improvements

There is a new Bacula database format (schema) in this version of Bacula that eliminates the FileName table by placing the Filename into the File record of the File table. This substantiallly improves performance, particularly for large databases.

The `update_xxx_catalog` script will automatically update the Bacula database format, but you should realize that for very large databases (greater than 50GB), it may take some time and it will double the size of the database on disk during the migration.

This database format change can provide very significant improvements in the speed of metadata insertion into the database, and in some cases (backup of large email servers) can significantly reduce the size of the database.

3.2 Automatic TLS Encryption

Starting with Bacula 11.0, all daemons and consoles are now using TLS automatically for all network communications. It is no longer required to setup TLS keys in advance. It is possible to turn off automatic TLS PSK encryption using the **TLS PSK Enable** directive.

3.3 Client Behind NAT Support with the Connect To Director Directive

A Client can now initiate a connection to the Director (permanently or scheduled) to allow the Director to communicate to the Client when a new Job is started or a bconsole command such as `status client` or `estimate` is issued.

This new network configuration option is particularly useful for Clients that are not directly reachable by the Director.

```
# cat /opt/bacula/etc/bacula-fd.conf
Director {
  Name = bac-dir
  Password = aigh3wu7oothieb4geeph3noo  # Password used to connect

  # New directives
  Address = bac-dir.mycompany.com       # Director address to connect
```

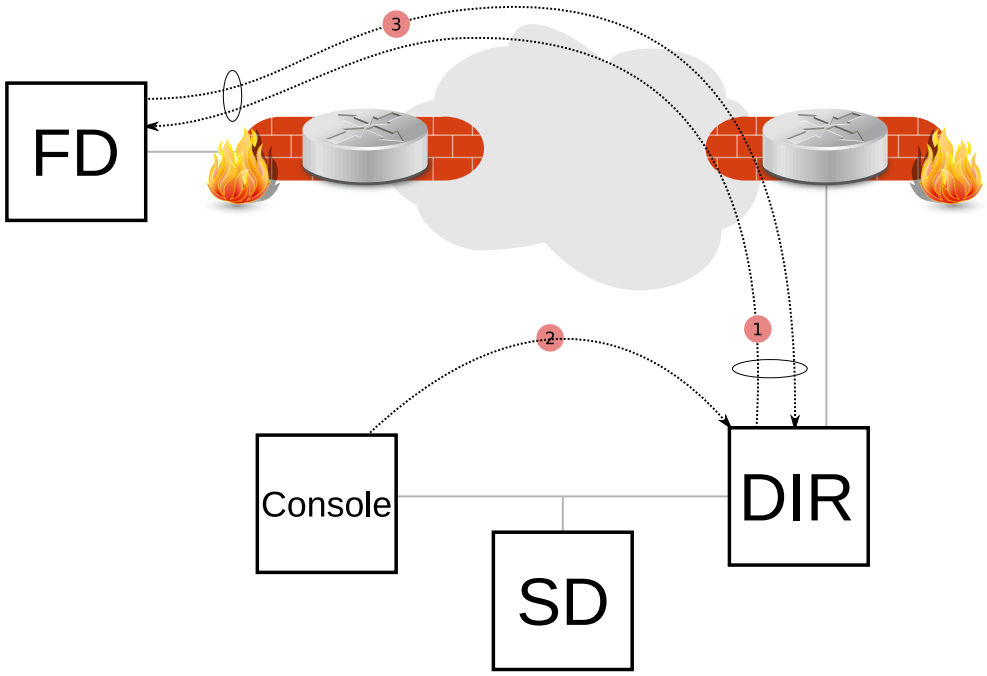


Figure 3.1: Client Behind NAT Example

```
  Connect To Director = yes                  # FD will call the Director
}


# cat /opt/bacula/etc/bacula-dir.conf
Client {
  Name = bac-fd
  Password = aigh3wu7oothieb4geeph3noo

  # New directive
  Allow FD Connections = yes
}
```

It is possible to schedule the Client connenction at certain periods of the day:

```
# cat /opt/bacula/etc/bacula-fd.conf
Director {
  Name = bac-dir
  Password = aigh3wu7oothieb4geeph3noo  # Password used to connect

  # New directives
  Address = bac-dir.mycompany.com       # Director address to connect
  Connect To Director = yes             # FD will call the Director
  Schedule = WorkingHours
}

Schedule {
  Name = WorkingHours
  # Connect the Director between 12:00 and 14:00
  Connect = MaxConnectTime=2h on mon-fri at 12:00
}
```

Note that in the current version, if the File Daemon is started after 12:00, the next connection to the Director will occur at 12:00 the next day.

A Job can be scheduled in the Director around 12:00, and if the Client is connected, the Job will be executed as if the Client was reachable from the Director.

3.4 Continuous Data Protection Plugin

Continuous Data Protection (CDP), also called continuous backup or real-time backup, refers to backup of Client data by automatically saving a copy of every change made to that data, essentially capturing every version of the data that the user saves. It allows the user or administrator to restore data to any point in time.

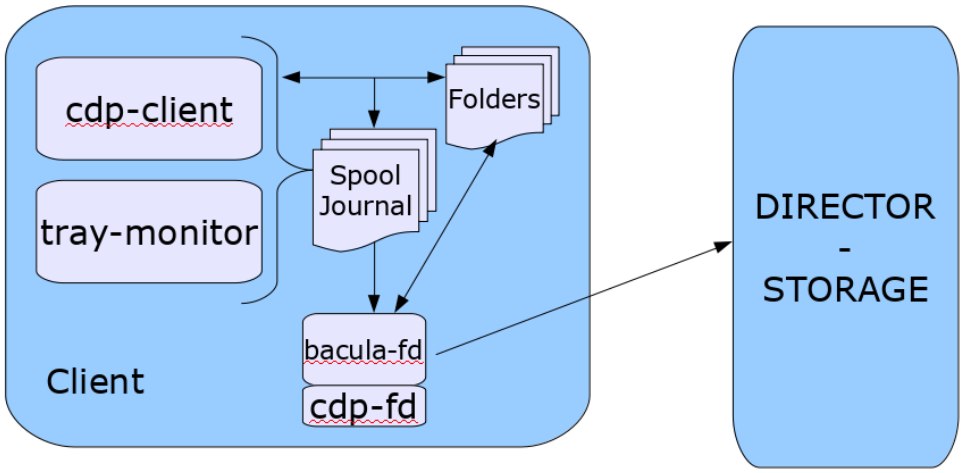


Figure 3.2: CDP Example

The Bacula CDP feature is composed of two components: An application (`cdp-client` or `tray-monitor`) that will monitor a set of directories configured by the user, and a Bacula FileDaemon plugin responsible to secure the data using Bacula infrastructure.

The user application (`cdp-client` or `tray-monitor`) is responsible for monitoring files and directories. When a modification is detected, the new data is copied into a *spool* directory. At a regular interval, a Bacula backup job will contact the FileDaemon and will save all the files archived by the `cdp-client`. The locally copied data can be restored at any time without a network connection to the Director.

See the CDP (Continious Data Protection) chapter 47 on page 509 for more information.

Global Autoprune Control Directive

The `Director` **Autoprune** directive can now globally control the Autoprune feature. This directive will take precedence over `Pool` or `Client` **Autoprune** directives.

```
Director {
  Name = mydir-dir
  ...
  AutoPrune = no     # switch off Autoprune globally
}
```

3.5 Event and Auditing

The Director daemon can now record events such as:

- Console connection/disconnection

- Daemon startup/shutdown
- Command execution
- ...

The events may be stored in a new catalog table, to disk, or sent via syslog.

```
Messages {
  Name = Standard
  catalog = all, events
  append = /opt/bacula/working/bacula.log = all, !skipped
  append = /opt/bacula/working/audit.log = events, !events.bweb
}

Messages {
  Name = Daemon
  catalog = all, events
  append = /opt/bacula/working/bacula.log = all, !skipped
  append = /opt/bacula/working/audit.log = events, !events.bweb
  append = /opt/bacula/working/bweb.log = events.bweb
}
```

The new message category "events" is not included in the default configuration files by default.

It is possible to filter out some events using "!events." form. It is possible to specify 10 custom events per Messages resource.

All event types are recorded by default.

When stored in the catalog, the events can be listed with the "list events" command.

```
 * list events [type=<str> | limit=<int> | order=<asc|desc> | days=<int> |
               start=<time-specification> | end=<time-specification>]
+---------------------+------------+-----------+--------------------------------+
| time                | type       | source    | event                          |
+---------------------+------------+-----------+--------------------------------+
| 2020-04-24 17:04:07 | daemon     | *Daemon*  | Director startup               |
| 2020-04-24 17:04:12 | connection | *Console* | Connection from 127.0.0.1:8101 |
| 2020-04-24 17:04:20 | command    | *Console* | purge jobid=1                  |
+---------------------+------------+-----------+--------------------------------+
```

The `.events` command can be used to record an external event. The source recorded will be recorded as "**source**". The events type can have a custom name.

```
 * .events type=baculum source=joe text="User login"
```

3.6 New Prune Command Option

The `prune jobs all` command will query the catalog to find all combinations of Client/Pool, and will run the pruning algorithm on each of them. At the end, all files and jobs not needed for restore that have passed the relevant retention times will be pruned.

The `prune` command `prune jobs all yes` can be scheduled in a RunScript to prune the catalog once per day for example. All Clients and Pools will be analyzed automatically.

```
Job {
  ...
  RunScript {
```

```
    Console = "prune jobs all yes"
    RunsWhen = Before
    failjobonerror = no
    runsonclient = no
  }
}
```

Dynamic Client Address Directive

It is now possible to use a script to determine the address of a Client when dynamic DNS option is not a viable solution:

```
Client {
  Name = my-fd
  ...
  Address = "|/opt/bacula/bin/compute-ip my-fd"
}
```

The command used to generate the address should return one single line with a valid address and end with the exit code 0. An example would be

```
  Address = "|echo 127.0.0.1"
```

This option might be useful in some complex cluster environments.

3.7 Volume Retention Enhancements

The Pool/Volume parameter `Volume Retention` can now be disabled to never prune a volume based on the `Volume Retention` time. When `Volume Retention` is disabled, only the `Job Retention` time will be used to prune jobs.

```
Pool {
  Volume Retention = 0
 ...
}
```

3.8 Windows Enhancements

- Support for Windows files with non-UTF16 names.
- Snapshot management has been improved, and a backup job now relies exclusively on the snapshot tree structure.
- Support for the `system.cifs_acl` extended attribute backup with Linux CIFS has been added. It can be used to backup Windows security attributes from a CIFS share mounted on a Linux system. Note that only recent Linux kernels can handle the `system.cifs_acl` feature correctly. The FileSet must use the `XATTR Support=yes` option, and the CIFS share must be mounted with the `cifsacl` options. See mount.cifs(8) for more information.

3.9 GPFS ACL Support

The new Bacula FileDaemon supports the GPFS filesystem specific ACL. The GPFS libraries must be installed in the standard location. To know if the GPFS support is available on your system, the following commands can be used.

```
*setdebug level=1 client=stretch-amd64-fd
Connecting to Client stretch-amd64-fd at stretch-amd64:9102
2000 OK setdebug=1 trace=0 hangup=0 blowup=0 options= tags=

*st client=stretch-amd64-fd
Connecting to Client stretch-amd64-fd at stretch-amd64:9102

stretch-amd64-fd Version: 11.0.0 (01 Dec 2020)  x86_64-pc-linux-gnu-bacula-enterprise debian 9.11
Daemon started 21-Jul-20 14:42. Jobs: run=0 running=0.
 Ulimits: nofile=1024 memlock=65536 status=ok
 Heap: heap=135,168 smbytes=199,993 max_bytes=200,010 bufs=104 max_bufs=105
 Sizes: boffset_t=8 size_t=8 debug=1 trace=0 mode=0,2010 bwlimit=0kB/s
 Crypto: fips=no crypto=OpenSSL 1.0.2u  20 Dec 2019
 APIs: GPFS
 Plugin: bpipe-fd.so(2)
```

The `APIs` line will indicate if the `/usr/lpp/mmfs/libgpfs.so` was loaded at the start of the Bacula FD service or not.

The standard `ACL Support` (cf 22.7 on page 253) directive can be used to enable automatically the support for the GPFS ACL backup.

3.10 New Baculum Features

3.10.1 Multi-user interface improvements

There have been added new functions and improvements to the multi-user interface and restricted access.

The Security page has new tabs:

- Console ACLs
- OAuth2 clients
- API hosts

These new tabs help to configure OAuth2 accounts, create restricted Bacula Console for users and create API hosts. They ease the process of creating users with a restricted Bacula resources access.

3.10.2 Add searching jobs by filename in the restore wizard

In the restore wizard now is possible to select job to restore by filename of file stored in backups. There is also possible to limit results to specific path.

3.10.3 Show more detailed job file list

The job file list now displays file details like: file attributes, UID, GID, size, mtime and information if the file record for saved or deleted file.

3.10.4 Add graphs to job view page

On the job view page, new pie and bar graphs for selected job are available.

3.10.5 Implement graphical status storage

On the storage status page are available two new types of the status (raw and graphical). The graphical status page is modern and refreshed asynchronously.

3.10.6 Add Russian translations

3.10.7 Global messages log window

There has been added new window to browse Bacula logs in a friendly way.

3.10.8 Job status weather

Add the job status weather on job list page to express current job condition.

3.10.9 Restore wizard improvements

In the restore wizard has been added listing and browsing names encoded in non-UTF encoding.

3.10.10 New API endpoints

- /oauth2/clients
- /oauth2/clients/client_id
- /jobs/files

3.10.11 New parameters in API endpoints

- /jobs/jobid/files - 'details' parameter
- /storages/show - 'output' parameter
- /storages/storageid/show - 'output' parameter

Chapter 4

New Features in 9.6.0

4.0.1 Building 9.6.4 and later

Version 9.6.4 is a major security and bug fix release. We suggest everyone to upgrade as soon as possible. <p> One significant improvement in this version is for the AWS S3 cloud driver. First the code base has been brought much closer to the Enterprise version (still a long ways to go). Second major change is that the community code now uses the latest version of libs3 as maintained by Bacula Systems. The libs3 code is available as a tar file for Bacula version 9.6.4 at: <p>

http://www.bacula.org/downloads/libs3-20200523.tar.gz

<p> Note: Version 9.6.4 must be compiled with the above libs3 version or later. To build libs3:

```
- Remove any libs3 package loaded by your OS
- Download above link
- tar xvfz libs3-20200523.tar.gz
- cd libs3-20200523
- make  # should have no errors
- sudo make install
```

<p> Then when you do your Bacula ./configure <args> it should automatically detect and use the libs3. The output from the ./configure will show whether or not libs3 was found during the configuration. E.g. <p>

```
   S3 support:                yes
```

<p> in the output from ./configure.

4.0.2 Docker Plugin

Containers is a relatively new system level virtualization concept that has less overhead than traditional virtualation. This is true because Container use the underlying operating system to provide all the needed services thus eliminating the need for multiple operating systems.

Docker containers rely on sophisticated file system level data abstraction with a number of read-only images to create templates used for container initialization.

With its Docker Plugin, the Bacula will save the full container image including all read-only and writable layers into a single image archive.

It is not necessary to install a Bacula File daemon in each container, so each container can be backed up from a common image repository.

The Bacula Docker Plugin will contact the Docker service to read and save the contents of any system image or container image using snapshots (default behavior) and dump them using the Docker API.

The Docker Plugin whitepaper provides more detailed information.

4.0.3 Real-Time Statistics Monitoring

All Bacula daemons can now collect internal performance statistics periodically and provide mechanisms to store the values to a CSV file or to send the values to a Graphite daemon via the network. Graphite is an enterprise-ready monitoring tool (https://graphiteapp.org).

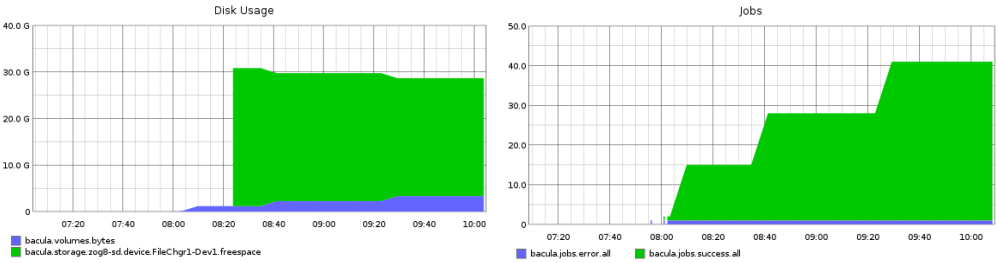


Figure 4.1: Graphite Examples

To activate the statistic collector feature, simply define a **Statistics** resource in the daemon of your choice:

```
Statistics {
 Name = "Graphite"
 Type = Graphite

# Graphite host information
 Host = "localhost"
 Port = 2003
}
```

It is possible to change the interval that is used to collect the statistics with the **Interval** directive (**5 mins** by default), and use the **Metrics** directive to select the data to collect (**all** by default).

If the Graphite daemon cannot be reached, the statistics data are spooled on disk and are sent automatically when the Graphite daemon is available again.

The bconsole **statistics** command can be used to display the current statistics in various formats (text or json for now).

```
*statistics
Statistics available for:
     1: Director
     2: Storage
     3: Client
Select daemon type for statistics (1-3): 1
bacula.dir.config.clients=1
```

```
bacula.dir.config.jobs=3
bacula.dir.config.filesets=2
bacula.dir.config.pools=3
bacula.dir.config.schedules=2
...
*statistics storage
...
bacula.storage.bac-sd.device.File1.readbytes=214
bacula.storage.bac-sd.device.File1.readtime=12
bacula.storage.bac-sd.device.File1.readspeed=0.000000
bacula.storage.bac-sd.device.File1.writespeed=0.000000
bacula.storage.bac-sd.device.File1.status=1
bacula.storage.bac-sd.device.File1.writebytes=83013529
bacula.storage.bac-sd.device.File1.writetime=20356
...
```

The **statistics** bconsole command can accept parameters to be scripted, for example it is possible to export the data in JSON, or to select which metrics to display.

```
*statistics bacula.dir.config.clients bacula.dir.config.jobs json
[
  {
    "name": "bacula.dir.config.clients",
    "value": 1,
    "type": "Integer",
    "unit": "Clients",
    "description": "The number of defined clients in the Director."
  },
  {
    "name": "bacula.dir.config.jobs",
    "value": 3,
    "type": "Integer",
    "unit": "Jobs",
    "description": "The number of defined jobs in the Director."
  }
]
```

The **.status statistics** command can be used to query the status of the Statistic collector thread.

```
*.status dir statistics
Statistics backend: Graphite is running
 type=2 lasttimestamp=12-Sep-18 09:45
 interval=300 secs
 spooling=in progress
 lasterror=Could not connect to localhost:2003 Err=Connection refused

Update Statistics: running interval=300 secs lastupdate=12-Sep-18 09:45
*
```

Chapter 5

New Features in 9.4.0

Cloud Backup

A major problem of Cloud backup is that data transmission to and from the Cloud is very slow compared to traditional backup to disk or tape. The Bacula Cloud drivers provide a means to quickly finish the backups and then to transfer the data from the local cache to the Cloud in the background. This is done by first splitting the data Volumes into small parts that are cached locally then uploading those parts to the Cloud storage service in the background, either while the job continues to run or after the backup Job has terminated. Once the parts are written to the Cloud, they may either be left in the local cache for quick restores or they can be removed (truncate cache).

Cloud Volume Architecture

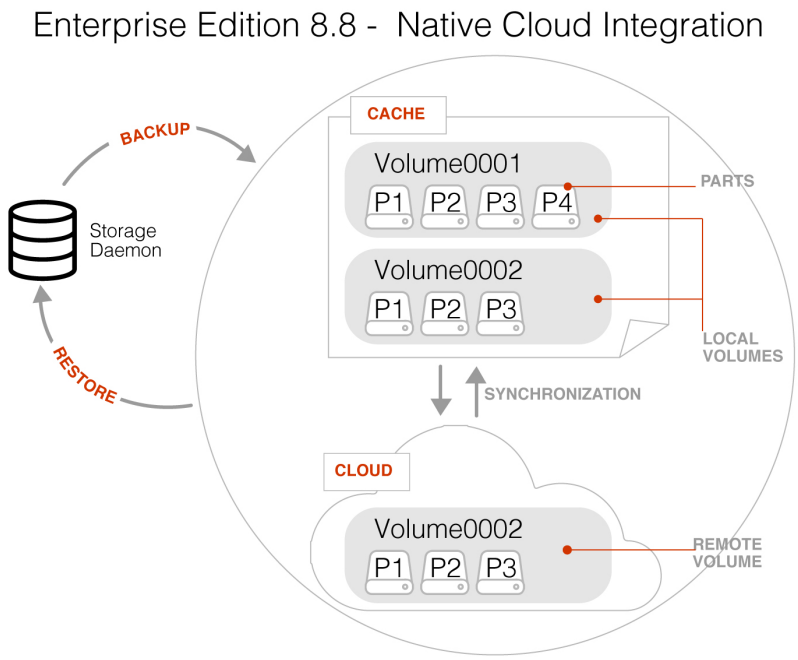


Figure 5.1: Bacula Cloud Architecture

The picture shown above shows two Volumes (Volume0001 and Volume0002) with their parts in the cache. Below the cache, one can see that Volume0002 has been uploaded or synchronized with the Cloud.

Note: Regular Bacula disk Volumes are implemented as standard files that reside in the user defined **Archive Directory**. On the other hand, Bacula Cloud Volumes are directories that reside in the user defined **Archive Directory**. Each Cloud Volume's directory contains the cloud Volume parts which are implemented as numbered files (part.1, part.2, ...).

Cloud Restore

During a restore, if the needed parts are in the local cache, they will be immediately used, otherwise, they will be downloaded from the Cloud as needed. The restore starts with parts already in the local cache but will wait in turn for any part that needs to be downloaded. The Cloud part downloads proceed while the restore is running.

With most Cloud providers, uploads are usually free of charge, but downloads of data from the Cloud are billed. By using local cache and multiple small parts, you can configure Bacula to substantially reduce download costs.

The `MaximumFileSize` Device directive is still valid within the Storage Daemon and defines the granularity of a restore chunk. In order to limit volume parts to download during restore (specially when restoring single files), it might be useful to set the `MaximumFileSize` to a value smaller than or equal to the `MaximumPartSize`.

Compatibility

Since a Cloud Volume contains the same data as an ordinary Bacula Volume, all existing types of Bacula data may be stored in the cloud – that is client encrypted, compressed data, plugin data, etc. All existing Bacula functionality, with the exception of deduplication, is available with the Bacula Cloud drivers.

Deduplication and the Cloud

At the current time, Bacula Global Endpoint Backup does not support writing to the cloud because the cloud would be too slow to support large hashed and indexed containers of deduplication data.

Virtual Autochangers and Disk Autochangers

If you use a Bacula Virtual Autochanger you will find it compatible with the new Bacula Cloud drivers. However, if you use a third party disk autochanger script such as **Vchanger**, unless or until it is modified to handle Volume directories, it may not be compatible with Bacula Cloud drivers.

Security

All data that is sent to and received from the cloud by default uses the HTTPS protocol, so your data is encrypted while being transmitted and received. However, data that resides in the Cloud is not encrypted by default. If you wish extra security of your data while it resides in the cloud, you should consider using Bacula's PKI data encryption feature during the backup.

Cache and Pruning

The Cache is treated much like a normal Disk based backup, so that in configuring Cloud the administrator should take care to set "Archive Device" in the Device resource to a directory where he/she would normally start data backed up to disk. Obviously, unless he/she uses the truncate/prune cache commands, the Archive Device will continue to fill.

The cache retention can be controlled per Volume with the "CacheRetention" attribute. The default value is 0, meaning that the pruning of the cache is disabled.

The "CacheRetention" value for a volume can be modified with the "update" command or via the Pool directive "CacheRetention" for newly created volumes.

5.0.1 New Commands, Resource, and Directives for Cloud

To support Cloud, there are new **bconsole** commands, new Storage Daemon directives and a new Cloud resource that is specified in the Storage Daemon's Device resource.

New Cloud Bconsole Commands

- **Cloud** The new cloud **bconsole** command allows you to do a number of things with cloud volumes. The options are the following:
  - None. If you specify no arguments to the command, bconsole will prompt with:

```
Cloud choice:
   1: List Cloud Volumes in the Cloud
   2: Upload a Volume to the Cloud
   3: Prune the Cloud Cache
   4: Truncate a Volume Cache
   5: Done
Select action to perform on Cloud (1-5):
```

    The different choices should be rather obvious.
  - **Truncate** This command will attempt to truncate the local cache for the specified Volume. Bacula will prompt you for the information needed to determine the Volume name or names. To avoid the prompts, the following additional command line options may be specified:
    - **Storage=xxx**
    - **Volume=xxx**
    - **AllPools**
    - **AllFromPool**
    - **Pool=xxx**
    - **MediaType=xxx**
    - **Drive=xxx**
    - **Slots=nnn**
  - **Prune** This command will attempt to prune the local cache for the specified Volume. Bacula will respect the CacheRetention volume attribute to determine if the cache can be truncated or not. Only parts that are uploaded to the cloud will be deleted from the cache. Bacula will prompt you for the information needed to determine the Volume name or names. To avoid the prompts, the following additional command line options may be specified:
    - **Storage=xxx**

    - **Volume=xxx**
    - **AllPools**
    - **AllFromPool**
    - **Pool=xxx**
    - **MediaType=xxx**
    - **Drive=xxx**
    - **Slots=nnn**
  - **Upload** This command will attempt to upload the specified Volumes. It will prompt you for the information needed to determine the Volume name or names. To avoid the prompts, you may specify any of the following additional command line options:
    - **Storage=xxx**
    - **Volume=xxx**
    - **AllPools**
    - **AllFromPool**
    - **Pool=xxx**
    - **MediaType=xxx**
    - **Drive=xxx**
    - **Slots=nnn**
  - **List** This command will list volumes stored in the Cloud. If a volume name is specified, the command will list all parts for the given volume. To avoid the prompts, you may specify any of the following additional command line options:
    - **Storage=xxx**
    - **Volume=xxx**
    - **Storage=xxx**

Cloud Additions to the DIR Pool Resource

Within the **bacula-dir.conf** file each **Pool** resource there is an additional keyword **CacheRetention** that can be specified.

Cloud Additions to the SD Device Resource

Within the **bacula-sd.conf** file each **Device** resource there is an additional keyword **Cloud** that must be specified on the **Device Type** directive, and two new directives **Maximum Part Size** and **Cloud**.

New Cloud SD Device Directives

- **Device Type** The Device Type has been extended to include the new keyword **Cloud** to specify that the device supports cloud Volumes. Example:

```
Device Type = Cloud
```

- **Cloud** The new Cloud directive permits specification of a new Cloud Resource. As with other Bacula resource specifications, one specifies the name of the Cloud resource. Example:

```
Cloud = S3Cloud
```

- **Maximum Part Size** This directive allows one to specify the maximum size for each part. Smaller part sizes will reduce restore costs, but may require a small additional overhead to handle multiple parts. The maximum number of parts permitted in a Cloud Volume is 524,288. The maximum size of any given part is approximately 17.5TB.

Example Cloud Device Specification

An example of a Cloud Device Resource might be:

```
Device {
  Name = CloudStorage
  Device Type = Cloud
  Cloud = S3Cloud
  Archive Device = /opt/bacula/backups
  Maximum Part Size = 10000000
  Media Type = CloudType
  LabelMedia = yes
  Random Access = Yes;
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
}
```

As you can see from the above, the **Cloud** directive in the **Device** resource contains the name (**S3Cloud**) of the **Cloud** resource that is shown below. Note also the **Archive Device** is specified in the same manner as one would use for a File device. However, in place of containing files with Volume names, the archive device for the Cloud drivers will contain the local cache, which consists of directories with the Volume name; and these directories contain the parts associated with the particular Volume. So with the above Device resource, and the two cache Volumes shown in figure **??** on page ?? above would have the following layout on disk:

```
/opt/bacula/backups
   /opt/bacula/backups/Volume0001
      /opt/bacula/backups/Volume0001/part.1
      /opt/bacula/backups/Volume0001/part.2
      /opt/bacula/backups/Volume0001/part.3
      /opt/bacula/backups/Volume0001/part.4
   /opt/bacula/backups/Volume0002
      /opt/bacula/backups/Volume0002/part.1
      /opt/bacula/backups/Volume0002/part.2
      /opt/bacula/backups/Volume0002/part.3
```

The Cloud Resource

The **Cloud** resource has a number of directives that may be specified as exemplified in the following example:

default east USA location:

```
Cloud {
  Name = S3Cloud
  Driver = "S3"
  HostName = "s3.amazonaws.com"
  BucketName = "BaculaVolumes"
  AccessKey = "BZIXAIS39DP9YNER5DFZ"
```

```
  SecretKey = "beesheeg7iTe0Gaexee7aedie4aWohfuewohGaa0"
  Protocol = HTTPS
  UriStyle = VirtualHost
  Truncate Cache = No
  Upload = EachPart
  Region = "us-east-1"
  MaximumUploadBandwidth = 5MB/s
}
```

central europe location:

```
Cloud {
  Name = S3Cloud
  Driver = "S3"
  HostName = "s3-eu-central-1.amazonaws.com"
  BucketName = "BaculaVolumes"
  AccessKey = "BZIXAIS39DP9YNER5DFZ"
  SecretKey = "beesheeg7iTe0Gaexee7aedie4aWohfuewohGaa0"
  Protocol = HTTPS
  UriStyle = VirtualHost
  Truncate Cache = No
  Upload = EachPart
  Region = "eu-central-1"
  MaximumUploadBandwidth = 4MB/s
}
```

For Amazon Cloud, refer to http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region to get a complete list of regions and corresponding endpoints and use them respectively as Region and HostName directive.

For CEPH S3 interface:

```
Cloud {
  Name = CEPH_S3
  Driver = "S3"
  HostName = ceph.mydomain.lan
  BucketName = "CEPHBucket"
  AccessKey = "xxxXXXxxxx"
  SecretKey = "xxheeg7iTe0Gaexee7aedie4aWohfuewohxx0"
  Protocol = HTTPS
  Upload = EachPart

  UriStyle = Path            # Must be set for CEPH
}
```

The directives of the above Cloud resource for the S3 driver are defined as follows:

Name = <Device-Name> The name of the Cloud resource. This is the logical Cloud name, and may be any string up to 127 characters in length. Shown as **S3Cloud** above.

Description = <Text> The description is used for display purposes as is the case with all resource.

Driver = <DriverName> This defines which driver to use. It can be **S3**. There is also a **File** driver, which is used mostly for testing.

Host Name = <Name> This directive specifies the hostname to be used in the URL. Each Cloud service provider has a different and unique hostname. The maximum size is 255 characters and may contain a tcp port specification.

Bucket Name = <Name> This directive specifies the bucket name that you wish to use on the Cloud service. This name is normally a unique name name that identifies where you want to place your Cloud Volume parts. With Amazon S3, the bucket must be created previously on the Cloud service. The maximum bucket name size is 255 characters.

Access Key = <String> The access key is your unique user identifier given to you by your cloud service provider.

Secret Key = <String> The secret key is the security key that was given to you by your cloud service provider. It is equivalent to a password.

Protocol = <HTTP | HTTPS> The protocol defines the communications protocol to use with the cloud service provider. The two protocols currently supported are: **HTTPS** and **HTTP**. The default is **HTTPS**.

Uri Style = <VirtualHost | Path> This directive specifies the URI style to use to communicate with the cloud service provider. The two Uri Styles currently supported are: **VirtualHost** and **Path**. The default is **VirtualHost**.

Truncate Cache = <Truncate-kw> This directive specifies when Bacula should automatically remove (truncate) the local cache parts. Local cache parts can only be removed if they have been uploaded to the cloud. The currently implemented values are:

- **No** Do not remove cache. With this option you must manually delete the cache parts with a **bconsole Truncate Cache** command, or do so with an **Admin** Job that runs an **Truncate Cache** command. This is the default.
- **AfterUpload** Each part will be removed just after it is uploaded. Note, if this option is specified, all restores will require a download from the Cloud. **N**ote: Not yet implemented.
- **AtEndOfJob** With this option, at the end of the Job, every part that has been uploaded to the Cloud will be removed (truncated). **N**ote: Not yet implemented.

Upload = <Upload-kw> This directive specifies when local cache parts will be uploaded to the Cloud. The options are:

- **No** Do not upload cache parts. With this option you must manually upload the cache parts with a **bconsole Upload** command, or do so with an **Admin** Job that runs an **Upload** command. This is the default.
- **EachPart** With this option, each part will be uploaded when it is complete i.e. when the next part is created or at the end of the Job.
- **AtEndOfJob** With this option all parts that have not been previously uploaded will be uploaded at the end of the Job. **N**ote: Not yet implemented.

Maximum Upload Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the upload bandwidth used globally by all devices referencing this Cloud resource.

Maximum Download Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the download bandwidth used globally by all devices referencing this Cloud resource.

Region = <String< The Cloud resource can be configured to use a specific endpoint within a region. This directive is required for AWS-V4 regions. ex: `Region="eu-central-1"`

File Driver for the Cloud

As mentioned above, one may specify the keyword **File** on the **Driver** directive of the Cloud resource. Instead of writing to the Cloud, Bacula will instead create a Cloud Volume but write it to disk. The rest of this section applies to the Cloud resource directives when the File driver is specified.

The following Cloud directives are ignored: Bucket Name, Access Key, Secret Key, Protocol, Uri Style. The directives **Truncate Cache** and **Upload** work on the local cache in the same manner as they do for the S3 driver. The main difference to note is that the Host Name, specifies the destination directory for the Cloud Volume files, and this Host Name must be different from the Archive Device name, or there will be a conflict between the local cache (in the Archive Device directory) and the destination Cloud Volumes (in the Host Name directory). As noted above, the File driver is mostly used for testing purposes, and we do not particularly recommend using it. However, if you have a particularly slow backup device you might want to stage your backup data into an SSD or disk using the local cache feature of the Cloud device, and have your Volumes transferred in the background to a slow File device.

5.0.2 WORM Tape Support

Automatic WORM (Write Once Read Multiple) tapes detection has been added in 10.2.

When a WORM tape is detected, the catalog volume entry is changed automatically to set `Recycle=no`. It will prevent the volume from being automatically recycled by Bacula.

There is no change in how the Job and File records are pruned from the catalog as that is a separate issue that is currently adequately implemented in Bacula.

When a WORM tape is detected, the SD will show WORM on the device state output (must have debug greater or equal to 6) otherwise the status shows as `!WORM`

```
Device state:
   OPENED !TAPE LABEL APPEND !READ !EOT !WEOT !EOF WORM !SHORT !MOUNTED ...
```

The output of the used volume status has been modified to include the worm state. It shows `worm=1` for a worm cassette and `worm=0` otherwise. Example:

```
Used Volume status:
Reserved volume: TestVolume001 on Tape device "nst0" (/dev/nst0)
   Reader=0 writers=0 reserves=0 volinuse=0 worm=1
```

The following programs are needed for the WORM tape detection:

- sdparm
- tapeinfo

The new Storage Device directive `Worm Command` must be configured as well as the `Control Device` directive (used with the Tape Alert feature).

```
Device {
  Name = "LTO-0"
  Archive Device = "/dev/nst0"
  Control Device = "/dev/sg0"    # from lsscsi -g
  Worm Command = "/opt/bacula/scripts/isworm %l"
...
}
```

Chapter 6

New Features in 9.2.0

This chapter describes new features that have been added to the current version of Bacula in version 9.2.0

In general, this is a fairly substantial release because it contains a very large number of bug fixes backported from the Bacula Enterprise version. There are also a few new features backported from Bacula Enterprise.

6.0.1 Enhanced Autochanger Support

Note: this feature was actually backported into version 9.0.0, but the documentation was added much after the 9.0.0 release. To call your attention to this new feature, we have also included the documentation here.

To make Bacula function properly with multiple Autochanger definitions, in the Director's configuration, you must adapt your **bacula-dir.conf Storage** directives.

Each autochanger that you have defined in an **Autochanger** resource in the Storage daemon's **bacula-sd.conf** file, must have a corresponding **Autochanger** resource defined in the Director's **bacula-dir.conf** file. Normally you will already have a **Storage** resource that points to the Storage daemon's **Autochanger** resource. Thus you need only to change the name of the **Storage** resource to **Autochanger**. In addition the **Autochanger = yes** directive is not needed in the Director's **Autochanger** resource, since the resource name is **Autochanger**, the Director already knows that it represents an autochanger.

In addition to the above change (**Storage** to **Autochanger**), you must modify any additional **Storage** resources that correspond to devices that are part of the **Autochanger** device. Instead of the previous **Autochanger = yes** directive they should be modified to be **Autochanger = xxx** where you replace the **xxx** with the name of the Autochanger.

For example, in the bacula-dir.conf file:

```
Autochanger {             # New resource
  Name = Changer-1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO-Changer-1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 50
}
```

```
Storage {
  Name = Changer-1-Drive0
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive0
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

Storage {
  Name = Changer-1-Drive1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

...
```

Note that Storage resources **Changer-1-Drive0** and **Changer-1-Drive1** are not required since they make up part of an autochanger, and normally, Jobs refer only to the Autochanger resource. However, by referring to those Storage definitions in a Job, you will use only the indicated drive. This is not normally what you want to do, but it is very useful and often used for reserving a drive for restores. See the Storage daemon example .conf below and the use of **AutoSelect = no**.

So, in summary, the changes are:

- Change **Storage** to **Autochanger** in the LTO4 resource.
- Remove the **Autochanger = yes** from the **Autochanger** LTO4 resource.
- Change the **Autochanger = yes** in each of the **Storage** device that belong to the **Autochanger** to point to the **Autochanger** resource with for the example above the directive **Autochanger = LTO4**.

  Please note that if you define two different autochangers, you must give a unique Media Type to the Volumes in each autochanger. More specifically, you may have multiple Media Types, but you cannot have Volumes with the same Media Type in two different autochangers. If you attempt to do so, Bacula will most likely reference the wrong autochanger (Storage) and not find the correct Volume.

6.0.2 New Prune Command Option

The bconsole "prune" command can now run the pruning algorithm on all volumes from a Pool or on all Pools.

```
* prune allfrompool pool=Default yes
* prune allfrompool allpools yes
```

6.0.3 BConsole Features

Delete a Client

The `delete client` bconsole command delete the database record of a client that is no longer defined in the configuration file. It also removes all other records (Jobs, Files, ...) associated with the client that is deleted.

Status Schedule Enhancements

The `status schedule` command can now accept multiple `client` or `job` keywords on the command line. The `limit` parameter is disabled when the `days` parameter is used. The output is now ordered by day.

Restore option "noautoparent"

During a `bconsole` restore session, parent directories are automatically selected to avoid issues with permissions. It is possible to disable this feature with the `noautoparent` command line parameter.

6.0.4 Tray Monitor Restore Screen

It is now possible to restore files from the Tray Monitor GUI program.

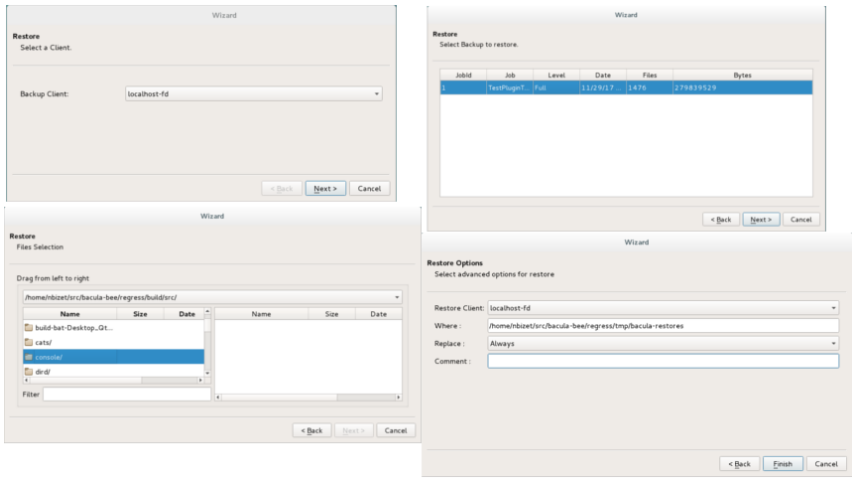


Figure 6.1: Tray Monitor Restore Wizard

Chapter 7

New Features in 9.0.0

This chapter describes new features that have been added to the current version of Bacula in version 9.0.0

7.0.1 Enhanced Autochanger Support

To make Bacula function properly with multiple Autochanger definitions, in the Director's configuration, you must adapt your **bacula-dir.conf Storage** directives.

Each autochanger that you have defined in an **Autochanger** resource in the Storage daemon's **bacula-sd.conf** file, must have a corresponding **Autochanger** resource defined in the Director's **bacula-dir.conf** file. Normally you will already have a **Storage** resource that points to the Storage daemon's **Autochanger** resource. Thus you need only to change the name of the **Storage** resource to **Autochanger**. In addition the **Autochanger = yes** directive is not needed in the Director's **Autochanger** resource, since the resource name is **Autochanger**, the Director already knows that it represents an autochanger.

In addition to the above change (**Storage** to **Autochanger**), you must modify any additional **Storage** resources that correspond to devices that are part of the **Autochanger** device. Instead of the previous **Autochanger = yes** directive they should be modified to be **Autochanger = xxx** where you replace the **xxx** with the name of the Autochanger.

For example, in the bacula-dir.conf file:

```
Autochanger {             # New resource
  Name = Changer-1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO-Changer-1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 50
}

Storage {
  Name = Changer-1-Drive0
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive0
  Media Type = LTO-4
```

```
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

Storage {
  Name = Changer-1-Drive1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

...
```

Note that Storage resources **Changer-1-Drive0** and **Changer-1-Drive1** are not required since they make up part of an autochanger, and normally, Jobs refer only to the Autochanger resource. However, by referring to those Storage definitions in a Job, you will use only the indicated drive. This is not normally what you want to do, but it is very useful and often used for reserving a drive for restores. See the Storage daemon example .conf below and the use of **AutoSelect = no**.

So, in summary, the changes are:

- Change **Storage** to **Autochanger** in the LTO4 resource.
- Remove the **Autochanger = yes** from the **Autochanger** LTO4 resource.
- Change the **Autochanger = yes** in each of the **Storage** device that belong to the **Autochanger** to point to the **Autochanger** resource with for the example above the directive **Autochanger = LTO4**.

7.0.2 Source Code for Windows

With this version of Bacula, we have included the old source code for Windows and also updated it to contain the code from the latest Bacula Enterprise version. The project is also directly distributing binaries for Windows rather than relying on Bacula Systems to supply them.

7.0.3 Maximum Virtual Full Interval Option

Two new director directives have been added:

```
   Max Virtual Full Interval
and
   Virtual Full Backup Pool
```

The **Max Virtual Full Interval** directive should behave similar to the **Max Full Interval**, but for Virtual Full jobs. If Bacula sees that there has not been a Full backup in Max Virtual Full Interval time then it will upgrade the job to Virtual Full. If you have both **Max Full Interval** and **Max Virtual Full Interval** set then Max Full Interval should take precedence.

The **Virtual Full Backup Pool** directive allows one to change the pool as well. You probably want to use these two directives in conjunction with each other but that may depend on the specifics of one's setup. If you set the **Max Full Interval** without setting **Max Virtual Full Interval** then Bacula will use whatever the "default" pool is set to which is the same behavior as with the Max Full Interval.

7.0.4 Progressive Virtual Full

In Bacula version 9.0.0, we have added a new Directive named **Backups To Keep** that permits you to implement Progressive Virtual Fulls within Bacula. Sometimes this feature is known as Incremental Forever with Consolidation.

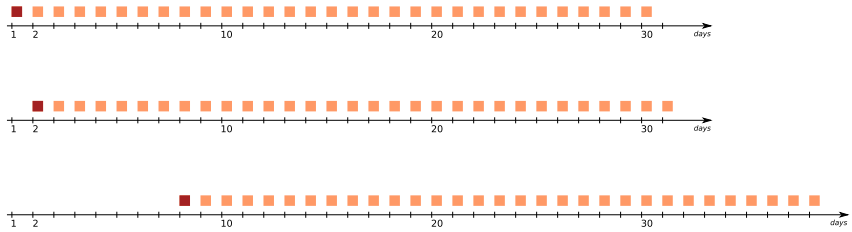


Figure 7.1: Backup Sequence Slides Forward One Day, Each Day

To implement the Progressive Virtual Full feature, simply add the **Backups To Keep** directive to your Virtual Full backup Job resource. The value specified on the directive indicates the number of backup jobs that should not be merged into the Virtual Full (i.e. the number of backup jobs that should remain after the Virtual Full has completed. The default is zero, which reverts to a standard Virtual Full than consolidates all the backup jobs that it finds.

Backups To Keep Directive

The new **BackupsToKeep** directive is specified in the Job Resource and has the form:

```
Backups To Keep = 30
```

where the value (30 in the above figure and example) is the number of backups to retain. When this directive is present during a Virtual Full (it is ignored for other Job types), it will look for the most recent Full backup that has more subsequent backups than the value specified. In the above example the Job will simply terminate unless there is a Full back followed by at least 31 backups of either level Differential or Incremental.

Assuming that the last Full backup is followed by 32 Incremental backups, a Virtual Full will be run that consolidates the Full with the first two Incrementals that were run after the Full. The result is that you will end up with a Full followed by 30 Incremental backups. The Job Resource in **bacula-dir.conf** to accomplish this would be:

```
Job {
  Name = "VFull"
  Type = Backup
  Level = VirtualFull
  Client = "my-fd"
  File Set = "FullSet"
  Accurate = Yes
  Backups To Keep = 10
}
```

Delete Consolidated Jobs

The new directive **Delete Consolidated Jobs** expects a **yes** or **no** value that if set to **yes** will cause any old Job that is consolidated during a Virtual Full to be deleted. In the example above we saw that a Full plus one other job (either an Incremental or Differential) were consolidated

into a new Full backup. The original Full plus the other Job consolidated will be deleted. The default value is **no**.

Virtual Full Compatibility

Virtual Full as well as Progressive Virtual Full works with any standard backup Job.

However, it should be noted that Virtual Full jobs are not compatible with any plugins that you may be using.

7.0.5 TapeAlert Enhancements

There are some significant enhancements to the TapeAlert feature of Bacula. Several directives are used slightly differently, which unfortunately causes a compatibility problem with the old TapeAlert implementation. Consequently, if you are already using TapeAlert, you must modify your **bacula-sd.conf** in order for Tape Alerts to work. See below for the details ...

What is New

First, you must define a **Alert Command** directive in the Device resource that calls the new **tapealert** script that is installed in the scripts directory (normally: /opt/bacula/scripts). It is defined as follows:

```
Device {
  Name = ...
  Archive Device = /dev/nst0
  Alert Command = "/opt/bacula/scripts/tapealert %l"
  Control Device = /dev/sg1 # must be SCSI ctl for /dev/nst0
  ...
}
```

In addition the **Control Device** directive in the Storage Daemon's conf file must be specified in each Device resource to permit Bacula to detect tape alerts on a specific devices (normally only tape devices).

Once the above mentioned two directives (Alert Command and Control Device) are in place in each of your Device resources, Bacula will check for tape alerts at two points:

- After the Drive is used and it becomes idle.
- After each read or write error on the drive.

At each of the above times, Bacula will call the new **tapealert** script, which uses the **tapeinfo** program. The tapeinfo utility is part of the apt sg3-utils and rpm sg3_utils packages that must be installed on your systems. Then after each alert that Bacula finds for that drive, Bacula will emit a Job message that is either INFO, WARNING, or FATAL depending on the designation in the Tape Alert published by the T10 Technical Committee on SCSI Storage Interfaces (www.t10.org). For the specification, please see: www.t10.org/ftp/t10/document.02/02-142r0.pdf

As a somewhat extreme example, if tape alerts 3, 5, and 39 are set, you will get the following output in your backup job.

```
17-Nov 13:37 rufus-sd JobId 1: Error: block.c:287
```

```
Write error at 0:17 on device "tape"
(/home/kern/bacula/k/regress/working/ach/drive0)
Vol=TestVolume001. ERR=Input/output error.

17-Nov 13:37 rufus-sd JobId 1: Fatal error: Alert:
Volume="TestVolume001" alert=3: ERR=The operation has stopped because
an error has occurred while reading or writing data which the drive
cannot correct.  The drive had a hard read or write error

17-Nov 13:37 rufus-sd JobId 1: Fatal error: Alert:
Volume="TestVolume001" alert=5: ERR=The tape is damaged or the drive
is faulty.  Call the tape drive supplier helpline.  The drive can no
longer read data from the tape

17-Nov 13:37 rufus-sd JobId 1: Warning: Disabled Device "tape"
(/home/kern/bacula/k/regress/working/ach/drive0) due to tape alert=39.

17-Nov 13:37 rufus-sd JobId 1: Warning: Alert: Volume="TestVolume001"
alert=39: ERR=The tape drive may have a fault.  Check for availability
of diagnostic information and run extended diagnostics if applicable.
The drive may have had a failure which may be identified by stored
diagnostic information or by running extended diagnostics (eg Send
Diagnostic).  Check the tape drive users manual for instructions on
running extended diagnostic tests and retrieving diagnostic data.
```

Without the tape alert feature enabled, you would only get the first error message above, which is the error return Bacula received when it gets the error. Notice also, that in the above output the alert number 5 is a critical error, which causes two things to happen. First the tape drive is disabled, and second the Job is failed.

If you attempt to run another Job using the Device that has been disabled, you will get a message similar to the following:

```
17-Nov 15:08 rufus-sd JobId 2: Warning:
     Device "tape" requested by DIR is disabled.
```

and the Job may be failed if no other drive can be found.

Once the problem with the tape drive has been corrected, you can clear the tape alerts and re-enable the device with the Bacula bconsole command such as the following:

```
  enable Storage=Tape
```

Note, when you enable the device, the list of prior tape alerts for that drive will be discarded.

Since is is possible to miss tape alerts, Bacula maintains a temporary list of the last 8 alerts, and each time Bacula calls the **tapealert** script, it will keep up to 10 alert status codes. Normally there will only be one or two alert errors for each call to the tapealert script.

Once a drive has one or more tape alerts, you can see them by using the bconsole status command as follows:

```
status storage=Tape
```

which produces the following output:

```
Device Vtape is "tape" (/home/kern/bacula/k/regress/working/ach/drive0)
mounted with:
    Volume:      TestVolume001
    Pool:        Default
    Media type:  tape
    Device is disabled. User command.
    Total Bytes Read=0 Blocks Read=1 Bytes/block=0
    Positioned at File=1 Block=0
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Hard Error
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Read Failure
    Warning Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Diagnostics Required
```

if you want to see the long message associated with each of the alerts, simply set the debug level to 10 or more and re-issue the status command:

```
setdebug storage=Tape level=10
status storage=Tape


    ...
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
      flags=0x0 alert=The operation has stopped because an error has occurred
       while reading or writing data which the drive cannot correct. The drive had
       a hard read or write error
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       flags=0x0 alert=The tape is damaged or the drive is faulty. Call the tape
       drive supplier helpline.  The drive can no longer read data from the tape
    Warning Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001" flags=0x1
       alert=The tape drive may have a fault. Check for availability of diagnostic
       information and run extended diagnostics if applicable.   The drive may
       have had a failure which may be identified by stored diagnostic information
       or by running extended diagnostics (eg Send Diagnostic). Check the tape
       drive users manual for instructions on running extended diagnostic tests
       and retrieving diagnostic data.
    ...
```

The next time you **enable** the Device by either using **bconsole** or you restart the Storage Daemon, all the saved alert messages will be discarded.

Handling of Alerts

Tape Alerts numbered 7,8,13,14,20,22,52,53, and 54 will cause Bacula to disable the current Volume.

Tape Alerts numbered 14,20,29,30,31,38, and 39 will cause Bacula to disable the drive.

Please note certain tape alerts such as 14 have multiple effects (disable the Volume and disable the drive).

7.0.6 New Console ACL Directives

By default, if a Console ACL directive is not set, Bacula will assume that the ACL list is empty. If the current Bacula Director configuration uses restricted Consoles and allows restore jobs, it is mandatory to configure the new directives.

DirectoryACL

This directive is used to specify a list of directories that can be accessed by a restore session. Without this directive, a restricted console cannot restore any file. Multiple directories names may be specified by separating them with commas, and/or by specifying multiple DirectoryACL directives. For example, the directive may be specified as:

```
DirectoryACL = /home/bacula/, "/etc/", "/home/test/*"
```

With the above specification, the console can access the following directories:

- /etc/password
- /etc/group
- /home/bacula/.bashrc
- /home/test/.ssh/config
- /home/test/Desktop/Images/something.png

But not to the following files or directories:

- /etc/security/limits.conf
- /home/bacula/.ssh/id_dsa.pub
- /home/guest/something
- /usr/bin/make

If a directory starts with a Windows pattern (ex: c:/), Bacula will automatically ignore the case when checking directory names.

7.0.7 New Bconsole "list" Command Behavior

The bconsole `list` commands can now be used safely from a restricted bconsole session. The information displayed will respect the ACL configured for the Console session. For example, if a restricted Console has access to JobA, JobB and JobC, information about JobD will not appear in the `list jobs` command.

7.0.8 New Console ACL Directives

It is now possible to configure a restricted Console to distinguish Backup and Restore job permissions. The `BackupClientACL` can restrict backup jobs on a specific set of clients, while the `RestoreClientACL` can restrict restore jobs.

```
# cat /opt/bacula/etc/bacula-dir.conf
...

Console {
 Name = fd-cons             # Name of the FD Console
 Password = yyy
...
 ClientACL = localhost-fd           # everything allowed
 RestoreClientACL = test-fd         # restore only
 BackupClientACL = production-fd    # backup only
}
```

The ClientACL directive takes precedence over the RestoreClientACL and the BackupClientACL. In the Console resource resource above, it means that the bconsole linked to the Console named "fd-cons" will be able to run:

- backup and restore for “localhost-fd”
- backup for “production-fd”
- restore for “test-fd”

At the restore time, jobs for client “localhost-fd”, “test-fd” and “production-fd” will be available.

If *all* is set for ClientACL, backup and restore will be allowed for all clients, despite the use of RestoreClientACL or "BackupClientACL.

7.0.9 Client Initiated Backup

A console program such as the new tray-monitor or bconsole can now be configured to connect a File Daemon. There are many new features available (see the New Tray Monitor section below), but probably the most important is the ability for the user to initiate a backup of his own machine. The connection established by the FD to the Director for the backup will be used by the Director for the backup, thus not only can clients (users) initiate backups, but a File Daemon that is NATed (cannot be reached by the Director) can now be backed up without using advanced tunneling techniques providing that the File Daemon can connect to the Director.

The flow of information is shown in the picture below:

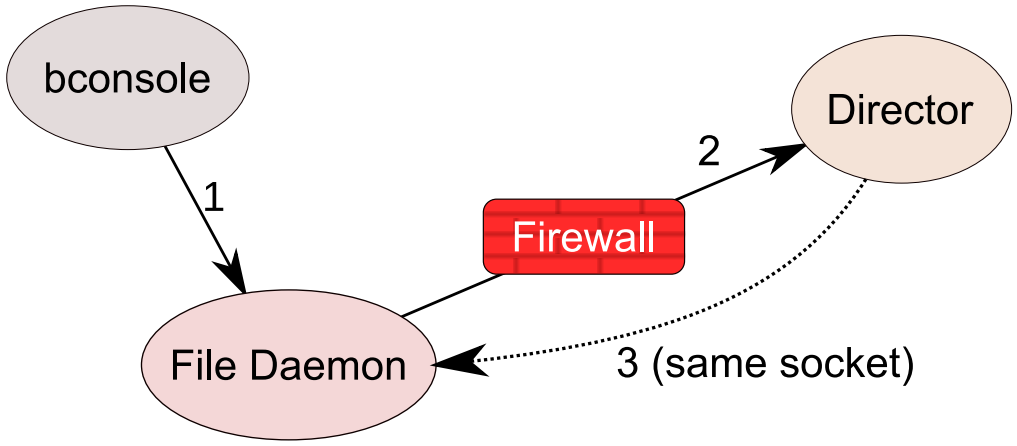


Figure 7.2: Client Initiated Backup Network Flow

7.0.10 Configuring Client Initiated Backup

In order to ensure security, there are a number of new directives that must be enabled in the new `tray-monitor`, the File Daemon and in the Director. A typical configuration might look like the following:

```
# cat /opt/bacula/etc/bacula-dir.conf
...

Console {
 Name = fd-cons              # Name of the FD Console
 Password = yyy

 # These commands are used by the tray-monitor, it is possible to restrict
 CommandACL = run, restore, wait, .status, .jobs, .clients
 CommandACL = .storages, .pools, .filesets, .defaults, .estimate

 # Adapt for your needs
 jobacl = *all*
 poolacl = *all*
 clientacl = *all*
 storageacl = *all*
 catalogacl = *all*
 filesetacl = *all*
}


# cat /opt/bacula/etc/bacula-fd.conf
...

Console {              # Console to connect the Director
  Name = fd-cons
  DIRPort = 9101
  address = localhost
  Password = "yyy"
}

Director {
  Name = remote-cons   # Name of the tray monitor/bconsole
  Password = "xxx"     # Password of the tray monitor/bconsole
  Remote = yes         # Allow to use send commands to the Console defined
}


cat /opt/bacula/etc/bconsole-remote.conf
....

Director {
  Name = localhost-fd
  address = localhost        # Specify the FD address
  DIRport = 9102             # Specify the FD Port
  Password = "notused"
}

Console {
  Name = remote-cons         # Name used in the auth process
  Password = "xxx"
}


cat ~/.bacula-tray-monitor.conf
Monitor {
  Name = remote-cons
}
```

```
Client {
  Name = localhost-fd
  address = localhost     # Specify the FD address
  Port = 9102             # Specify the FD Port
  Password = "xxx"
  Remote = yes
}
```

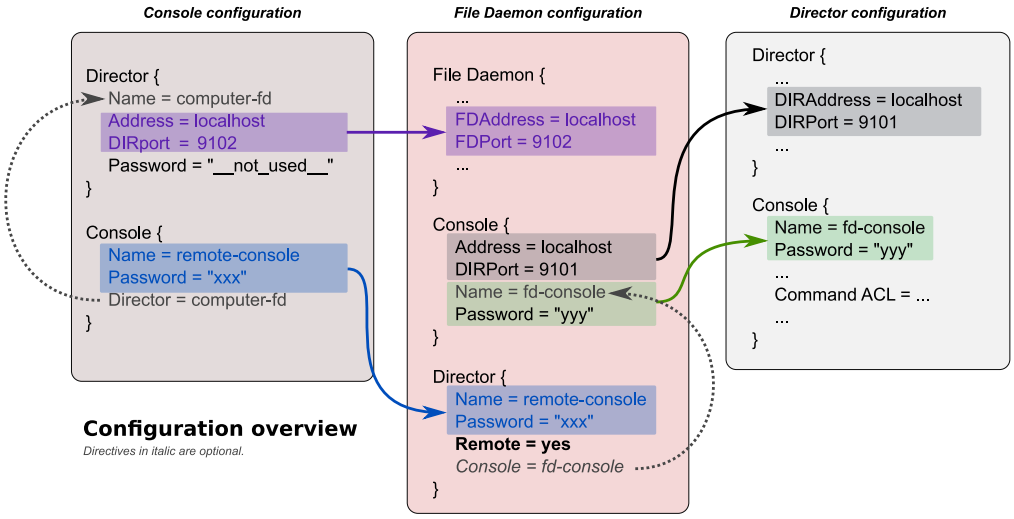


Figure 7.3: Relation Between Resources (bconsole)

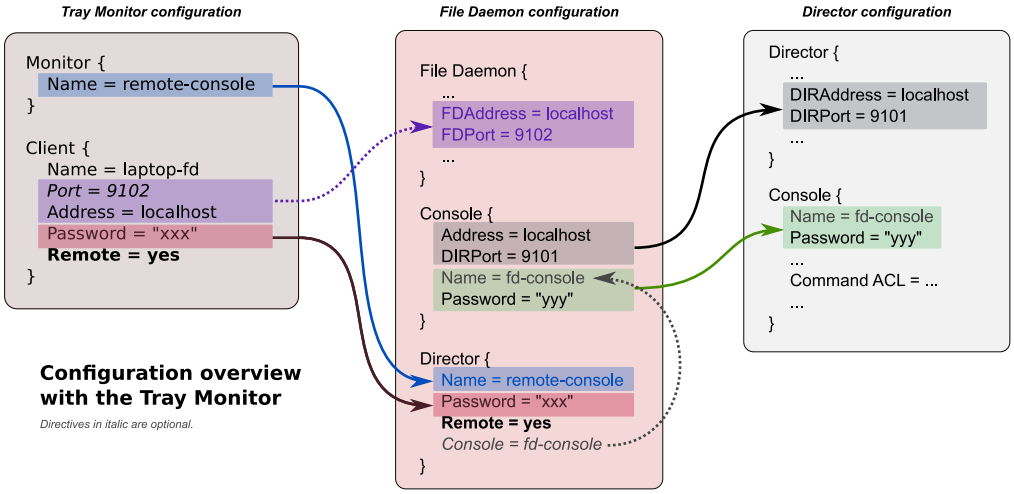


Figure 7.4: Relation Between Resources (tray-monitor)

A more detailed description with complete examples is available in chapter **??**.

7.0.11 New Tray Monitor

A new tray monitor has been added to the 9.0 release, the tray monitor offers the following features:

- Director, File and Storage Daemon status page

- Support for the Client Initiated Backup protocol (See 7.0.9 on page 52). To use the Client Initiated Backup option from the tray monitor, the Client option "Remote" should be checked in the configuration (Fig 54.7 on page 601).
- Wizard to run new job (Fig 54.9 on page 602)
- Display an estimation of the number of files and the size of the next backup job (Fig 54.9 on page 602)
- Ability to configure the tray monitor configuration file directly from the GUI (Fig 54.7 on page 601)
- Ability to monitor a component and adapt the tray monitor task bar icon if a jobs are running.
- TLS Support
- Better network connection handling
- Default configuration file is stored under `$HOME/.bacula-tray-monitor.conf`
- Ability to "schedule" jobs
- Available on Linux and Windows platforms

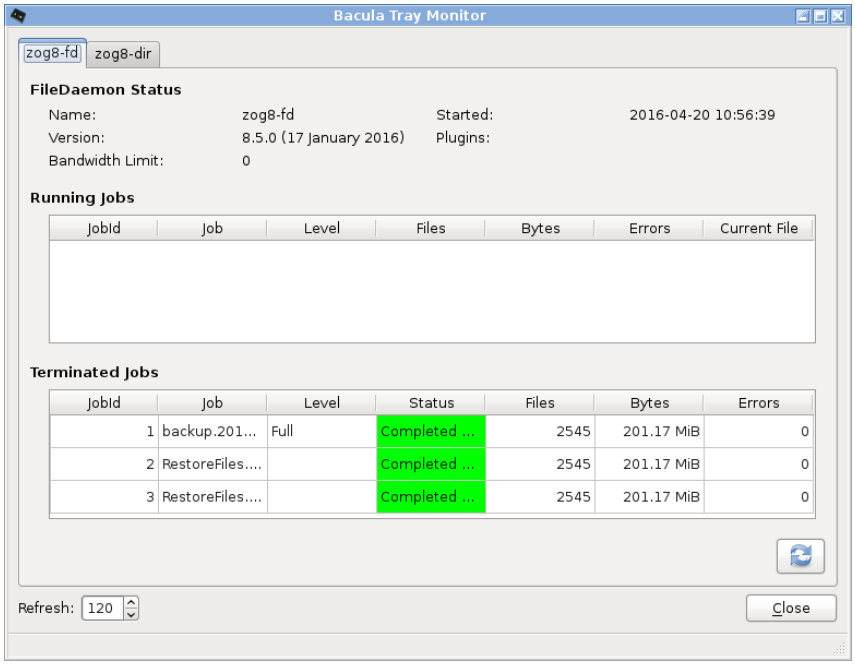


Figure 7.5: Tray Monitor Status

7.0.12 Schedule Jobs via the Tray Monitor

The Tray Monitor can scan periodically a specific directory "Command Directory" and process "*.bcmd" files to find jobs to run.

The format of the "file.bcmd" command file is the following:

```
<component name>:<run command>
<component name>:<run command>
...

<component name> = string
<run command>    = string (bconsole command line)
```

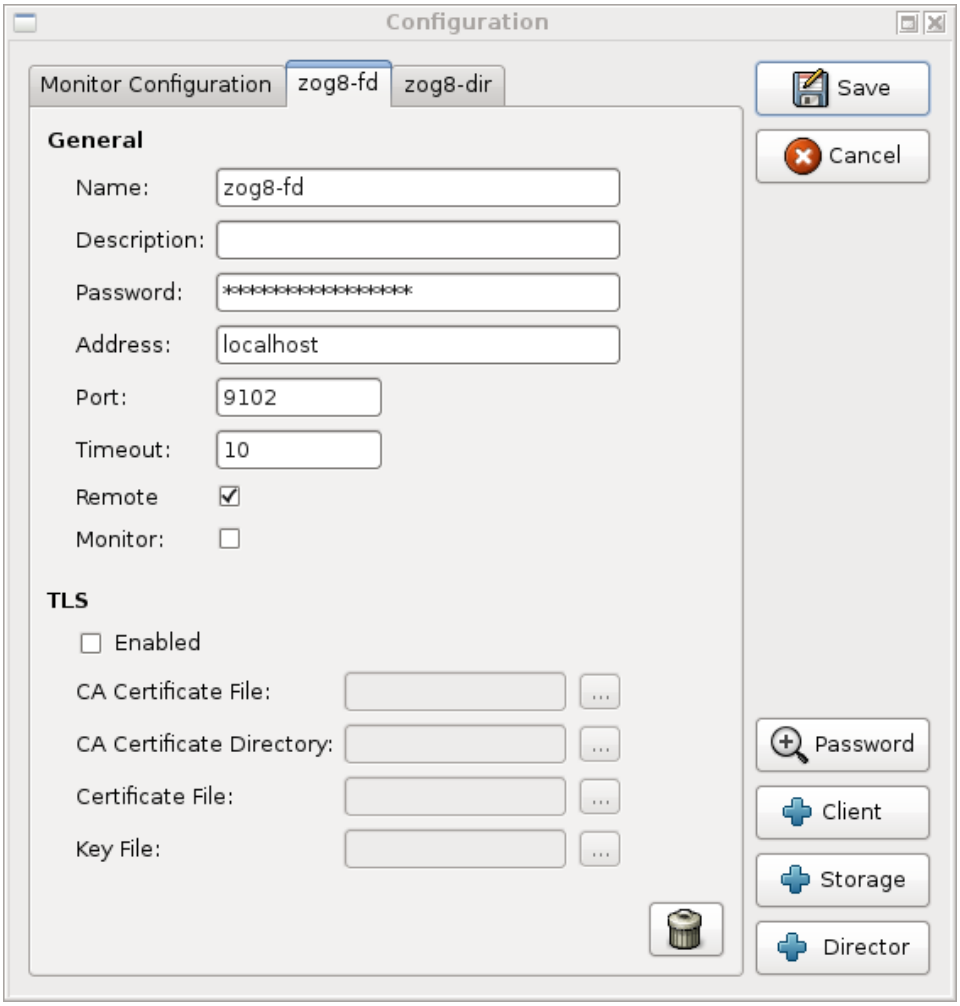


Figure 7.6: Tray Monitor Client Configuration

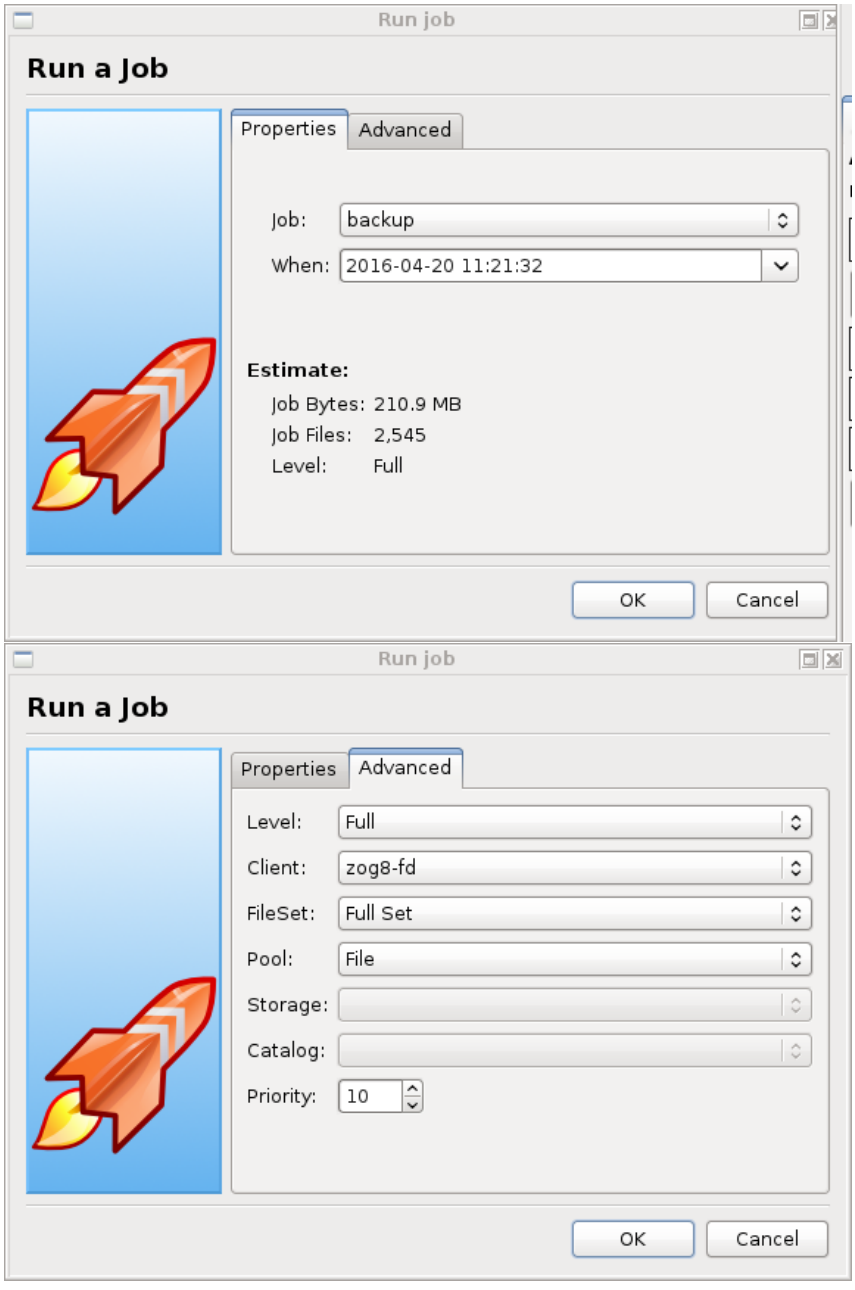


Figure 7.7: Tray Monitor Run a Job

For example:

```
localhost-fd: run job=backup-localhost-fd level=full
localhost-dir: run job=BackupCatalog
```

The command file should contain at least one command. The component specified in the first part of the command line should be defined in the tray monitor. Once the command file is detected by the tray monitor, a popup is displayed to the user and it is possible for the user to cancel the job directly.

The file can be created with tools such as "cron" or the "task scheduler" on Windows. It is possible to verify the network connection at that time to avoid network errors.

```
#!/bin/sh
if ping -c 1 director &> /dev/null
then
   echo "my-dir: run job=backup" > /path/to/commands/backup.bcmd
fi
```

7.0.13 Accurate Option for Verify "Volume Data" Job

Since Bacula version 8.4.1, it has been possible to have a Verify Job configured with `level=Data` that will reread all records from a job and optionally check the size and the checksum of all files. Starting with

Bacula version 9.0, it is now possible to use the `accurate` option to check catalog records at the same time. When using a Verify job with `level=Data` and `accurate=yes` can replace the `level=VolumeToCatalog` option.

For more information on how to setup a Verify Data job, see 54.5.1 on page 607.

To run a Verify Job with the `accurate` option, it is possible to set the option in the Job definition or set use the `accurate=yes` on the command line.

```
* run job=VerifyData jobid=10 accurate=yes
```

7.0.14 FileDaemon Saved Messages Resource Destination

It is now possible to send the list of all saved files to a Messages resource with the `saved` message type. It is not recommended to send this flow of information to the director and/or the catalog when the client FileSet is pretty large. To avoid side effects, the `all` keyword doesn't include the `saved` message type. The `saved` message type should be explicitly set.

```
# cat /opt/bacula/etc/bacula-fd.conf
...
Messages {
  Name = Standard
  director = mydirector-dir = all, !terminate, !restored, !saved
  append = /opt/bacula/working/bacula-fd.log = all, saved, restored
}
```

7.0.15 Minor Enhancements

New Bconsole ".estimate" Command

The new `.estimate` command can be used to get statistics about a job to run. The command uses the database to approximate the size and the number of files of the next job. On a PostgreSQL database, the command uses regression slope to compute values. On MySQL, where these statistical functions are not available, the command uses a simple "average" estimation. The correlation number is given for each value.

```
*.estimate job=backup
level=I
nbjob=0
corrbytes=0
jobbytes=0
corrfiles=0
jobfiles=0
duration=0
job=backup

*.estimate job=backup level=F
level=F
nbjob=1
corrbytes=0
jobbytes=210937774
corrfiles=0
jobfiles=2545
duration=0
job=backup
```

Traceback and Lockdump

After the reception of a signal, `traceback` and `lockdump` information are now stored in the same file.

7.0.16 Bconsole "list jobs" command options

The `list jobs` bconsole command now accepts new command line options:

- **joberrors** Display jobs with JobErrors
- **jobstatus=T** Display jobs with the specified status code
- **client=cli** Display jobs for a specified client
- **order=asc/desc** Change the output format of the job list. The jobs are sorted by start time and JobId, the sort can use ascendant (asc) or descendant (desc) (default) value.

7.0.17 Minor Enhancements

New Bconsole "Tee All" Command

The "@tall" command allows logging all input/output from a console session.

```
*@tall /tmp/log
*st dir
...
```

7.0.18 Bconsole "list jobs" command options

The `list jobs` bconsole command now accepts new command line options:

- **joberrors** Display jobs with JobErrors
- **jobstatus=T** Display jobs with the specified status code
- **client=cli** Display jobs for a specified client
- **order=asc/desc** Change the output format of the job list. The jobs are sorted by start time and JobId, the sort can use ascendant (asc) or descendant (desc) (default) value.

7.0.19 New Bconsole "Tee All" Command

The "@tall" command allows logging all input/output from a console session.

```
*@tall /tmp/log
*st dir
...
```

7.0.20 New Job Edit Codes %I

In various places such as RunScripts, you have now access to %I to get the JobId of the copy or migration job started by a migrate job.

```
Job {
  Name = Migrate-Job
  Type = Migrate
  ...
  RunAfter = "echo New JobId is %I"
}
```

.api version 2

In Bacula version 9.0 and later, we introduced a new .api version to help external tools to parse various Bacula bconsole output.

The `api_opts` option can use the following arguments:

C Clear current options

tn Use a specific time format (1 ISO format, 2 Unix Timestamp, 3 Default Bacula time format)

sn Use a specific separator between items (new line by default).

Sn Use a specific separator between objects (new line by default).

o Convert all keywords to lowercase and convert all non *isalpha* characters to _

```
  .api 2 api_opts=t1s43S35
  .status dir running
==================================
jobid=10
job=AJob
...
```

New Debug Options

In Bacula version 9.0 and later, we introduced a new `options` parameter for the `setdebug` bconsole command.

The following arguments to the new `option` parameter are available to control debug functions.

- 0 Clear debug flags
- i Turn off, ignore bwrite() errors on restore on File Daemon
- d Turn off decomp of BackupRead() streams on File Daemon
- t Turn on timestamps in traces
- T Turn off timestamps in traces
- c Truncate trace file if trace file is activated
- l Turn on recording events on P() and V()
- p Turn on the display of the event ring when doing a backtrace

The following command will enable debugging for the File Daemon, truncate an existing trace file, and turn on timestamps when writing to the trace file.

```
* setdebug level=10 trace=1 options=ct fd
```

It is now possible to use a *class* of debug messages called `tags` to control the debug output of Bacula daemons.

- all Display all debug messages
- bvfs Display BVFS debug messages
- sql Display SQL related debug messages
- memory Display memory and poolmem allocation messages
- scheduler Display scheduler related debug messages

```
* setdebug level=10 tags=bvfs,sql,memory
* setdebug level=10 tags=!bvfs

# bacula-dir -t -d 200,bvfs,sql
```

The `tags` option is composed of a list of tags. Tags are separated by "," or "+" or "-" or "!". To disable a specific tag, use "-" or "!" in front of the tag. Note that more tags are planned for future versions.

7.0.21 Communication Line Compression

Bacula version 9.0.0 and later now includes communication line compression. It is turned on by default, and if the two Bacula components (Dir, FD, SD, bconsole) are both version 6.6.0 or greater, communication line compression) will be enabled, by default. If for some reason, you do not want communication line compression, you may disable it with the following directive:

```
Comm Compression = no
```

This directive can appear in the following resources:

```
bacula-dir.conf: Director resource
bacula-fd.conf Client (or FileDaemon) resource
bacula-sd.conf: Storage resource
bconsole.conf: Console resource
bat.conf: Console resource
```

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled (default) In the case that the compression is not effective, Bacula turns it off on a. record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports **None** in the Job report.

7.0.22 Deduplication Optimized Volumes

This version of Bacula includes a new alternative (or additional) volume format that optimizes the placement of files so that an underlying deduplicating filesystem such as ZFS can optimally deduplicate the backup data that is written by Bacula. These are called Deduplication Optimized Volumes or Aligned Volumes for short. The details of how to use this feature and its considerations are in the Deduplication Optimized Volumes whitepaper.

This feature is available if you have Bacula Community produced binaries and the Aligned Volumes plugin.

7.0.23 baculabackupreport

I have added a new script called **baculabackupreport** to the scripts directory. This script was written by Bill Arlofski. It prints a backup summary of the backups that occurred in the prior number of hours specified on the command line. You need to edit the first few lines of the file to ensure that your email address is correct and the database type you are using is correct (default is PostgreSQL). Once you do that, you can manually implement it with:

```
/opt/bacula/scripts/baculabackupreport 24
```

I have put the above line in my **scripts/delete_catalog_backup** script so that it will be mailed to me nightly.

7.0.24 New Message Identification Format

We are starting to add unique message identifiers to each message (other than debug and the Job report) that Bacula prints. At the current time only two files in the Storage Daemon have these message identifiers and over time with subsequent releases we will modify all messages.

The message identifier will be kept unique for each message and once assigned to a message it will not change even if the text of the message changes. This means that the message identifier will be the same no matter what language the text is displayed in, and more importantly, it will allow us to make listing of the messages with in some cases, additional explanation or instructions on how to correct the problem. All this will take several years since it is a lot of work and requires some new programs that are not yet written to manage these message identifiers.

The format of the message identifier is:

```
[AAnnnn]
```

where A is an upper case character and nnnn is a four digit number, where the first character indicates the software component (daemon); the second letter indicates the severity, and the number is unique for a given component and severity.

For example:

```
[SF0001]
```

The first character representing the component at the current time one of the following:

```
S       Storage daemon
D       Director
F       File daemon
```

The second character representing the severity or level can be:

```
A       Abort
F       Fatal
E       Error
W       Warning
S       Security
I       Info
D       Debug
O       OK (i.e. operation completed normally)
```

So in the example above [SF0001] indicates it is a message id, because of the brackets and because it is at the beginning of the message, and that it was generated by the Storage daemon as a fatal error. As mentioned above it will take some time to implement these message ids everywhere, and over time we may add more component letters and more severity levels as needed.

Chapter 8

New Features in 7.4.0

This chapter presents the new features that have been added to the various versions of Bacula.

8.1 New Features in 7.4.3

8.1.1 RunScripts

There are two new RunScript short cut directives implemented in the Director. They are:

```
Job {
  ...
  ConsoleRunBeforeJob = "console-command"
  ...
}

Job {
  ...
  ConsoleRunAfterJob = "console-command"
  ...
}
```

As with other RunScript commands, you may have multiple copies of either the **ConsoleRunBeforeJob** or the **ConsoleRunAfterJob** in the same Job resource definition. Please note that not all console commands are permitted, and that if you run a console command that requires a response, the results are not determined (i.e. it will probably fail).

8.2 New Features in 7.4.0

8.2.1 Verify Volume Data

It is now possible to have a Verify Job configured with `level=Data` to reread all records from a job and optionally check the size and the checksum of all files.

```
# Verify Job definition
Job {
```

```
  Name = VerifyData
  Level = Data
  Client = 127.0.0.1-fd     # Use local file daemon
  FileSet = Dummy           # Will be adapted during the job
  Storage = File            # Should be the right one
  Messages = Standard
  Pool = Default
}

# Backup Job definition
Job {
  Name = MyBackupJob
  Type = Backup
  Client = windows1
  FileSet = MyFileSet
  Pool = 1Month
  Storage = File
}

FileSet {
  Name = MyFileSet
  Include {
    Options {
      Verify = s5
      Signature = MD5
    }
  File = /
}
```

To run the Verify job, it is possible to use the "jobid" parameter of the "run" command.

```
*run job=VerifyData jobid=10
Run Verify Job
JobName:     VerifyData
Level:       Data
Client:      127.0.0.1-fd
FileSet:     Dummy
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  MyBackupJob.2015-11-11_09.41.55_03
Verify List: /opt/bacula/working/working/VerifyVol.bsr
When:        2015-11-11 09:47:38
Priority:    10
OK to run? (yes/mod/no): yes
Job queued. JobId=14

...

11-Nov 09:46 my-dir JobId 13: Bacula 7.4.0 (13Nov15):
  Build OS:               x86_64-unknown-linux-gnu archlinux
  JobId:                  14
  Job:                    VerifyData.2015-11-11_09.46.29_03
  FileSet:                MyFileSet
  Verify Level:           Data
  Client:                 127.0.0.1-fd
  Verify JobId:           10
  Verify Job:q
  Start time:             11-Nov-2015 09:46:31
  End time:               11-Nov-2015 09:46:32
```

```
Files Expected:          1,116
Files Examined:          1,116
Non-fatal FD errors:     0
SD Errors:               0
FD termination status:   Verify differences
SD termination status:   OK
Termination:             Verify Differences
```

The current Verify Data implementation requires specifying the correct Storage resource in the Verify job. The Storage resource can be changed with the bconsole command line and with the menu.

8.2.2 Bconsole "list jobs" command options

The `list jobs` bconsole command now accepts new command line options:

- **joberrors** Display jobs with JobErrors
- **jobstatus=T** Display jobs with the specified status code
- **client=cli** Display jobs for a specified client
- **order=asc/desc** Change the output format of the job list. The jobs are sorted by start time and JobId, the sort can use ascendant (asc) or descendant (desc) (default) value.

8.2.3 Minor Enhancements

New Bconsole "Tee All" Command

The "@tall" command allows logging all input/output from a console session.

```
*@tall /tmp/log
*st dir
...
```

8.2.4 Windows Encrypted File System (EFS) Support

The Bacula Enterprise Windows File Daemon for the community version 7.4.0 now automatically supports files and directories that are encrypted on Windows filesystem.

8.2.5 SSL Connections to MySQL

There are five new Directives for the Catalog resource in the **bacula-dir.conf** file that you can use to encrypt the communications between Bacula and MySQL for additional security.

dbsslkey takes a string variable that specifies the filename of an SSL key file.

dbsslcert takes a string variable that specifies the filename of an SSL certificate file.

dbsslca takes a string variable that specifies the filename of a SSL CA (certificate authority) certificate.

dbsslcipher takes a string variable that specifies the cipher to be used.

8.2.6 Max Virtual Full Interval

This is a new Job resource directive that specifies the time in seconds that is a maximum time between Virtual Full jobs. It is much like the Max Full Interval directive but applies to Virtual Full jobs rather that Full jobs.

8.2.7 New List Volumes Output

The **list** and **llist** commands have been modified so that when listing Volumes a new pseudo field **expiresin** will be printed. This field is the number of seconds in which the retention period will expire. If the retention period has already expired the value will be zero. Any non-zero value means that the retention period is still in effect.

An example with many columns shorted for display purpose is:

```
*list volumes
Pool: Default
*list volumes
Pool: Default
+----+---------------+-----------+---------+-------------+-----------+
| id | volumename    | volstatus | enabled | volbytes    | expiresin |
+----+---------------+-----------+---------+-------------+-----------+
|  1 | TestVolume001 | Full      |       1 | 249,940,696 |         0 |
|  2 | TestVolume002 | Full      |       1 | 249,961,704 |         1 |
|  3 | TestVolume003 | Full      |       1 | 249,961,704 |         2 |
|  4 | TestVolume004 | Append    |       1 | 127,367,896 |         3 |
+----+---------------+-----------+---------+-------------+-----------+
```

Chapter 9

New Features in 7.2.0

This chapter presents the new features that have been added to the various versions of Bacula.

9.1 New Features in 7.2.0

9.1.1 New Job Edit Codes %E %R

In various places such as RunScripts, you have now access to %E to get the number of non-fatal errors for the current Job and %R to get the number of bytes read from disk or from the network during a job.

9.1.2 Enable/Disable commands

The **bconsole enable** and **disable** commands have been extended from enabling/disabling Jobs to include Clients, Schedule, and Storage devices. Examples:

```
disable Job=NightlyBackup Client=Windows-fd
```

will disable the Job named **NightlyBackup** as well as the client named **Windows-fd**.

```
disable Storage=LTO-changer Drive=1
```

will disable the first drive in the autochanger named **LTO-changer**.

Please note that doing a **reload** command will set any values changed by the enable/disable commands back to the values in the bacula-dir.conf file.

The Client and Schedule resources in the bacula-dir.conf file now permit the directive Enable = yes or Enable = no.

9.2 Bacula 7.2

9.2.1 Snapshot Management

Bacula 7.2 is now able to handle Snapshots on Linux/Unix systems. Snapshots can be automatically created and used to backup files. It is also possible to manage Snapshots from Bacula's `bconsole` tool through a unique interface.

Snapshot Backends

The following Snapshot backends are supported:

- BTRFS
- ZFS
- LVM[1]

By default, Snapshots are mounted (or directly available) under **.snapshots** directory on the root filesystem. (On ZFS, the default is **.zfs/snapshots**).

The Snapshot backend program is called **bsnapshot** and is available in the **bacula-enterprise-snapshot** package. In order to use the Snapshot Management feature, the package must be installed on the Client.

The **bsnapshot** program can be configured using `/opt/bacula/etc/bsnapshot.conf` file. The following parameters can be adjusted in the configuration file:

- `trace=<file>` Specify a trace file
- `debug=<num>` Specify a debug level
- `sudo=<yes/no>` Use sudo to run commands
- `disabled=<yes/no>` Disable snapshot support
- `retry=<num>` Configure the number of retries for some operations
- `snapshot_dir=<dirname>` Use a custom name for the Snapshot directory. (**.SNAPSHOT**, **.snapdir**, etc...)
- `lvm_snapshot_size=<lvpath:size>` Specify a custom snapshot size for a given LVM volume

```
# cat /opt/bacula/etc/bsnapshot.conf
trace=/tmp/snap.log
debug=10
lvm_snapshot_size=/dev/ubuntu-vg/root:5%
```

Application Quiescing

When using Snapshots, it is very important to quiesce applications that are running on the system. The simplest way to quiesce an application is to stop it. Usually, taking the Snapshot is very fast, and the downtime is only about a couple of seconds. If downtime is not possible and/or the application provides a way to quiesce, a more advanced script can be used. An example is described on 9.2.1 on the next page.

[1] Some restrictions described in 9.2.1 on page 73 applies to the LVM backend

New Director Directives

The use of the Snapshot Engine on the FileDaemon is determined by the new **Enable Snapshot** FileSet directive. The default is **no**.

```
FileSet {
  Name = LinuxHome

  Enable Snapshot = yes

  Include {
    Options = { Compression = LZO }
    File = /home
  }
}
```

By default, Snapshots are deleted from the Client at the end of the backup. To keep Snapshots on the Client and record them in the Catalog for a determined period, it is possible to use the **Snapshot Retention** directive in the Client or in the Job resource. The default value is 0 secconds. If, for a given Job, both Client and Job **Snapshot Retention** directives are set, the Job directive will be used.

```
Client {
   Name = linux1
   ...

   Snapshot Retention = 5 days
}
```

To automatically prune Snapshots, it is possible to use the following RunScript command:

```
Job {
   ...
   Client = linux1
   ...
   RunScript {
      RunsOnClient = no
      Console = "prune snapshot client=%c yes"
      RunsAfter = yes
   }
}
```

In RunScripts, the `AfterSnapshot` keyword for the `RunsWhen` directive will allow a command to be run just after the Snapshot creation. `AfterSnapshot` is a synonym for the `AfterVSS` keyword.

```
Job {
 ...
  RunScript {
    Command = "/etc/init.d/mysql start"
    RunsWhen = AfterSnapshot
    RunsOnClient = yes
  }
  RunScript {
    Command = "/etc/init.d/mysql stop"
```

```
    RunsWhen = Before
    RunsOnClient = yes
  }
}
```

Job Output Information

Information about Snapshots are displayed in the Job output. The list of all devices used by the Snapshot Engine is displayed, and the Job summary indicates if Snapshots were available.

```
JobId 3:    Create Snapshot of /home/build
JobId 3:    Create Snapshot of /home/build/subvol
JobId 3:    Delete snapshot of /home/build
JobId 3:    Delete snapshot of /home/build/subvol
...
JobId 3: Bacula 127.0.0.1-dir 7.2.0 (23Jul15):
  Build OS:               x86_64-unknown-linux-gnu archlinux
  JobId:                  3
  Job:                    Incremental.2015-02-24_11.20.27_08
  Backup Level:           Full
...
  Snapshot/VSS:           yes
...
  Termination:            Backup OK
```

New "snapshot" Bconsole Commands

The new **snapshot** command will display by default the following menu:

```
*snapshot
Snapshot choice:
     1: List snapshots in Catalog
     2: List snapshots on Client
     3: Prune snapshots
     4: Delete snapshot
     5: Update snapshot parameters
     6: Update catalog with Client snapshots
     7: Done
Select action to perform on Snapshot Engine (1-7):
```

The **snapshot** command can also have the following parameters:

```
[client=<client-name> | job=<job-name> | jobid=<jobid>]
 [delete | list | listclient | prune | sync | update]
```

It is also possible to use traditional `list`, `llist`, `update`, `prune` or `delete` commands on Snapshots.

```
*llist snapshot jobid=5
 snapshotid: 1
       name: NightlySave.2015-02-24_12.01.00_04
 createdate: 2015-02-24 12:01:03
     client: 127.0.0.1-fd
    fileset: Full Set
```

```
      jobid: 5
     volume: /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
     device: /home/btrfs
       type: btrfs
  retention: 30
    comment:


* snapshot listclient
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:8102
Snapshot      NightlySave.2015-02-24_12.01.00_04:
  Volume:     /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
  Device:     /home
  CreateDate: 2015-02-24 12:01:03
  Type:       btrfs
  Status:     OK
  Error:
```

With the *Update catalog with Client snapshots* option (or **snapshot sync**), the Director contacts the FileDaemon, lists snapshots of the system and creates catalog records of the Snapshots.

```
*snapshot sync
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:8102
Snapshot      NightlySave.2015-02-24_12.35.47_06:
  Volume:     /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
  Device:     /home
  CreateDate: 2015-02-24 12:35:47
  Type:       btrfs
  Status:     OK
  Error:
Snapshot added in Catalog

*llist snapshot
 snapshotid: 13
       name: NightlySave.2015-02-24_12.35.47_06
 createdate: 2015-02-24 12:35:47
     client: 127.0.0.1-fd
    fileset:
      jobid: 0
     volume: /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
     device: /home
       type: btrfs
  retention: 0
    comment:
```

LVM Backend Restrictions

LVM Snapshots are quite primitive compared to ZFS, BTRFS, NetApp and other systems. For example, it is not possible to use Snapshots if the Volume Group (VG) is full. The administrator must keep some free space in the VG to create Snapshots. The amount of free space required depends on the activity of the Logical Volume (LV). **bsnapshot** uses 10% of the LV by default. This number can be configured per LV in the **bsnapshot.conf** file.

```
[root@system1]# vgdisplay
  --- Volume group ---
```

```
  VG Name               vg_ssd
  System ID
  Format                lvm2
...
  VG Size               29,81 GiB
  PE Size               4,00 MiB
  Total PE              7632
  Alloc PE / Size       125 / 500,00 MiB
  Free  PE / Size       7507 / 29,32 GiB
...
```

It is also not advisable to leave snapshots on the LVM backend. Having multiple snapshots of the same LV on LVM will slow down the system.

Debug Options

To get low level information about the Snapshot Engine, the debug tag "snapshot" should be used in the **setdebug** command.

```
* setdebug level=10 tags=snapshot client
* setdebug level=10 tags=snapshot dir
```

9.2.2 Minor Enhancements

Storage Daemon Reports Disk Usage

The `status storage` command now reports the space available on disk devices:

```
...
Device status:

Device file: "FileStorage" (/bacula/arch1) is not open.
    Available Space=5.762 GB
==

Device file: "FileStorage1" (/bacula/arch2) is not open.
    Available Space=5.862 GB
```

9.2.3 Data Encryption Cipher Configuration

Bacula Enterprise version 8.0 and later now allows configuration of the data encryption cipher and the digest algorithm. Previously, the cipher was forced to AES 128, but it is now possible to choose between the following ciphers:

- AES128 (default)
- AES192
- AES256
- blowfish

The digest algorithm was set to SHA1 or SHA256 depending on the local OpenSSL options. We advise you to not modify the PkiDigest default setting. Please, refer to the OpenSSL documentation to understand the pros and cons regarding these options.

```
  FileDaemon {
    ...
    PkiCipher = AES256
  }
```

New Option Letter "M" for Accurate Directive in FileSet

Added in version 8.0.5, the new "M" option letter for the Accurate directive in the FileSet Options block, which allows comparing the modification time and/or creation time against the last backup timestamp. This is in contrast to the existing options letters "m" and/or "c", mtime and ctime, which are checked against the stored catalog values, which can vary accross different machines when using the BaseJob feature.

The advantage of the new "M" option letter for Jobs that refer to BaseJobs is that it will instruct Bacula to backup files based on the last backup time, which is more useful because the mtime/ctime timestamps may differ on various Clients, causing files to be needlessly backed up.

```
  Job {
    Name = USR
    Level = Base
    FileSet = BaseFS
...
  }

  Job {
    Name = Full
    FileSet = FullFS
    Base = USR
...
  }

  FileSet {
    Name = BaseFS
    Include {
      Options {
        Signature = MD5
      }
      File = /usr
    }
  }

  FileSet {
    Name = FullFS
    Include {
      Options {
        Accurate = Ms      # check for mtime/ctime of last backup timestamp and Size
        Signature = MD5
      }
      File = /home
      File = /usr
    }
  }
```

9.2.4 Read Only Storage Devices

This version of Bacula allows you to define a Storage deamon device to be read-only. If the **Read Only** directive is specified and enabled, the drive can only be used for read operations. The **Read Only** directive can be defined in any bacula-sd.conf Device resource, and is most useful for reserving one or more drives for restores. An example is:

```
Read Only = yes
```

9.2.5 New Resume Command

The new `resume` command does exactly the same thing as a **restart** command, but for some users the name may be more logical because in general the **restart** command is used to resume running a Job that was incomplete.

9.2.6 New Prune "Expired" Volume Command

In Bacula Enterprise 6.4, it is now possible to prune all volumes (from a pool, or globally) that are "expired". This option can be scheduled after or before the backup of the catalog and can be combined with the `Truncate On Purge` option. The `prune expired volme` command may be used instead of the `manual_prune.pl` script.

```
* prune expired volume

* prune expired volume pool=FullPool
```

To schedule this option automatically, it can be added to the Catalog backup job definition.

```
Job {
  Name = CatalogBackup
  ...
  RunScript {
    Console = "prune expired volume yes"
    RunsWhen = Before
  }
}
```

9.2.7 New Job Edit Codes %P %C

In various places such as RunScripts, you have now access to %P to get the current Bacula process ID (PID) and %C to know if the current job is a cloned job.

9.2.8 Enhanced Status and Error Messages

We have enhanced the Storage daemon status output to be more readable. This is important when there are a large number of devices. In addition to formatting changes, it also includes more details on which devices are reading and writing.

A number of error messages have been enhanced to have more specific data on what went wrong.

If a file changes size while being backed up the old and new size are reported.

9.2.9 Miscellaneous New Features

- Allow unlimited line lengths in .conf files (previously limited to 2000 characters).
- Allow /dev/null in ChangerCommand to indicated a Virtual Autochanger.
- Add a –fileprune option to the manual_prune.pl script.
- Add a -m option to make_catalog_backup.pl to do maintenance on the catalog.
- Safer code that cleans up the working directory when starting the daemons. It limits what files can be deleted, hence enhances security.
- Added a new .ls command in bconsole to permit browsing a client's filesystem.
- Fixed a number of bugs, includes some obscure seg faults, and a race condition that occurred infrequently when running Copy, Migration, or Virtual Full backups.
- Upgraded to a newer version of Qt4 for bat. All indications are that this will improve bat's stability on Windows machines.
- The Windows installers now detect and refuse to install on an OS that does not match the 32/64 bit value of the installer.

9.2.10 FD Storage Address

When the Director is behind a NAT, in a WAN area, to connect to the StorageDaemon, the Director uses an "external" ip address, and the FileDaemon should use an "internal" IP address to contact the StorageDaemon.

The normal way to handle this situation is to use a canonical name such as "storage-server" that will be resolved on the Director side as the WAN address and on the Client side as the LAN address. This is now possible to configure this parameter using the new directive `FDStorageAddress` in the Storage or Client resource.

```
Storage {
     Name = storage1
     Address = 65.1.1.1
     FD Storage Address = 10.0.0.1
     SD Port = 9103
     ...
}
```

```
 Client {
      Name = client1
      Address = 65.1.1.2
      FD Storage Address = 10.0.0.1
      FD Port = 9102
      ...
 }
```

Note that using the Client `FDStorageAddress` directive will not allow to use multiple Storage Daemon, all Backup or Restore requests will be sent to the specified `FDStorageAddress`.

9.2.11 Maximum Concurrent Read Jobs

This is a new directive that can be used in the **bacula-dir.conf** file in the Storage resource. The main purpose is to limit the number of concurrent Copy, Migration, and VirtualFull jobs so that they don't monopolize all the Storage drives causing a deadlock situation where all the drives

are allocated for reading but none remain for writing. This deadlock situation can occur when running multiple simultaneous Copy, Migration, and VirtualFull jobs.

The default value is set to 0 (zero), which means there is no limit on the number of read jobs. Note, limiting the read jobs does not apply to Restore jobs, which are normally started by hand. A reasonable value for this directive is one half the number of drives that the Storage resource has rounded down. Doing so, will leave the same number of drives for writing and will generally avoid over committing drives and a deadlock.

9.2.12 Incomplete Jobs

During a backup, if the Storage daemon experiences disconnection with the File daemon during backup (normally a comm line problem or possibly an FD failure), under conditions that the SD determines to be safe it will make the failed job as Incomplete rather than failed. This is done only if there is sufficient valid backup data that was written to the Volume. The advantage of an Incomplete job is that it can be restarted by the new bconsole **restart** command from the point where it left off rather than from the beginning of the jobs as is the case with a cancel.

9.2.13 The Stop Command

Bacula has been enhanced to provide a **stop** command, very similar to the **cancel** command with the main difference that the Job that is stopped is marked as Incomplete so that it can be restarted later by the **restart** command where it left off (see below). The **stop** command with no arguments, will like the cancel command, prompt you with the list of running jobs allowing you to select one, which might look like the following:

```
*stop
Select Job:
     1: JobId=3 Job=Incremental.2012-03-26_12.04.26_07
     2: JobId=4 Job=Incremental.2012-03-26_12.04.30_08
     3: JobId=5 Job=Incremental.2012-03-26_12.04.36_09
Choose Job to stop (1-3): 2
2001 Job "Incremental.2012-03-26_12.04.30_08" marked to be stopped.
3000 JobId=4 Job="Incremental.2012-03-26_12.04.30_08" marked to be stopped.
```

9.2.14 The Restart Command

The new **Restart command** allows console users to restart a canceled, failed, or incomplete Job. For canceled and failed Jobs, the Job will restart from the beginning. For incomplete Jobs the Job will restart at the point that it was stopped either by a stop command or by some recoverable failure.

If you enter the **restart** command in bconsole, you will get the following prompts:

```
*restart
You have the following choices:
     1: Incomplete
     2: Canceled
     3: Failed
     4: All
Select termination code:  (1-4):
```

If you select the **All** option, you may see something like:

```
Select termination code:  (1-4): 4
+-------+-------------+---------------------+------+-------+----------+-----------+---------
| jobid | name        | starttime           | type | level | jobfiles |
jobbytes  | jobstatus |
+-------+-------------+---------------------+------+-------+----------+-----------+---------
|     1 | Incremental | 2012-03-26 12:15:21 | B    | F     |        0 |
    0 | A         |
|     2 | Incremental | 2012-03-26 12:18:14 | B    | F     |      350 |
4,013,397 | I         |
|     3 | Incremental | 2012-03-26 12:18:30 | B    | F     |        0 |
    0 | A         |
|     4 | Incremental | 2012-03-26 12:18:38 | B    | F     |      331 |
3,548,058 | I         |
+-------+-------------+---------------------+------+-------+----------+-----------+---------
Enter the JobId list to select:
```

Then you may enter one or more JobIds to be restarted, which may take the form of a list of JobIds separated by commas, and/or JobId ranges such as **1-4**, which indicates you want to restart JobIds 1 through 4, inclusive.

9.2.15 Job Bandwidth Limitation

The new **Job Bandwidth Limitation** directive may be added to the File daemon's and/or Director's configuration to limit the bandwidth used by a Job on a Client. It can be set in the File daemon's conf file for all Jobs run in that File daemon, or it can be set for each Job in the Director's conf file. The speed is always specified in bytes per second.

For example:

```
FileDaemon {
  Name = localhost-fd
  Working Directory = /some/path
  Pid Directory = /some/path
  ...
  Maximum Bandwidth Per Job = 5Mb/s
}
```

The above example would cause any jobs running with the FileDaemon to not exceed 5 megabytes per second of throughput when sending data to the Storage Daemon. Note, the speed is always specified in bytes per second (not in bits per second), and the case (upper/lower) of the specification characters is ignored (i.e. 1MB/s = 1Mb/s).

You may specify the following speed parameter modifiers: k/s (1,000 bytes per second), kb/s (1,024 bytes per second), m/s (1,000,000 bytes per second), or mb/s (1,048,576 bytes per second).

For example:

```
Job {
  Name = locahost-data
  FileSet = FS_localhost
  Accurate = yes
  ...
  Maximum Bandwidth = 5Mb/s
  ...
}
```

The above example would cause Job `localhost-data` to not exceed 5MB/s of throughput when sending data from the File daemon to the Storage daemon.

A new console command `setbandwidth` permits to set dynamically the maximum throughput of a running Job or for future jobs of a Client.

```
* setbandwidth limit=1000 jobid=10
```

Please note that the value specified for the `limit` command line parameter is always in units of 1024 bytes (i.e. the number is multiplied by 1024 to give the number of bytes per second). As a consequence, the above limit of 1000 will be interpreted as a limit of 1000 * 1024 = 1,024,000 bytes per second.

9.2.16 Always Backup a File

When the Accurate mode is turned on, you can decide to always backup a file by using then new **A** Accurate option in your FileSet. For example:

```
Job {
   Name = ...
   FileSet = FS_Example
   Accurate = yes
   ...
}

FileSet {
 Name = FS_Example
 Include {
   Options {
     Accurate = A
   }
   File = /file
   File = /file2
 }
 ...
}
```

This project was funded by Bacula Systems based on an idea of James Harper and is available with the Bacula Enterprise Edition.

9.2.17 Setting Accurate Mode at Runtime

You are now able to specify the Accurate mode on the `run` command and in the Schedule resource.

```
* run accurate=yes job=Test
```

```
Schedule {
  Name = WeeklyCycle
  Run = Full 1st sun at 23:05
  Run = Differential accurate=yes 2nd-5th sun at 23:05
  Run = Incremental  accurate=no  mon-sat at 23:05
}
```

It can allow you to save memory and and CPU resources on the catalog server in some cases.

These advanced tuning options are available with the Bacula Enterprise Edition.

9.2.18 Additions to RunScript variables

You can have access to JobBytes, JobFiles and Director name using %b, %F and %D in your runscript command. The Client address is now available through %h.

```
RunAfterJob = "/bin/echo Job=%j JobBytes=%b JobFiles=%F ClientAddress=%h Dir=%D"
```

9.2.19 LZO Compression

LZO compression was added in the Unix File Daemon. From the user point of view, it works like the GZIP compression (just replace **compression=GZIP** with **compression=LZO**).

For example:

```
Include {
   Options { compression=LZO }
   File = /home
   File = /data
}
```

LZO provides much faster compression and decompression speed but lower compression ratio than GZIP. It is a good option when you backup to disk. For tape, the built-in compression may be a better option.

LZO is a good alternative for GZIP1 when you don't want to slow down your backup. On a modern CPU it should be able to run almost as fast as:

- your client can read data from disk. Unless you have very fast disks like SSD or large/fast RAID array.
- the data transfers between the file daemon and the storage daemon even on a 1Gb/s link.

Note that bacula only use one compression level LZO1X-1.

The code for this feature was contributed by Laurent Papier.

9.2.20 Purge Migration Job

The new **Purge Migration Job** directive may be added to the Migration Job definition in the Director's configuration file. When it is enabled the Job that was migrated during a migration will be purged at the end of the migration job.

For example:

```
Job {
  Name = "migrate-job"
  Type = Migrate
  Level = Full
  Client = localhost-fd
```

```
  FileSet = "Full Set"
  Messages = Standard
  Storage = DiskChanger
  Pool = Default
  Selection Type = Job
  Selection Pattern = ".*Save"
...
  Purge Migration Job = yes
}
```

This project was submitted by Dunlap Blake; testing and documentation was funded by Bacula Systems.

9.2.21 Changes in the Pruning Algorithm

We rewrote the job pruning algorithm in this version. Previously, in some users reported that the pruning process at the end of jobs was very long. It should not be longer the case. Now, Bacula won't prune automatically a Job if this particular Job is needed to restore data. Example:

```
JobId: 1  Level: Full
JobId: 2  Level: Incremental
JobId: 3  Level: Incremental
JobId: 4  Level: Differential
.. Other incrementals up to now
```

In this example, if the Job Retention defined in the Pool or in the Client resource causes that Jobs with Jobid in 1,2,3,4 can be pruned, Bacula will detect that JobId 1 and 4 are essential to restore data at the current state and will prune only JobId 2 and 3.

`Important`, this change affect only the automatic pruning step after a Job and the `prune jobs` Bconsole command. If a volume expires after the `VolumeRetention` period, important jobs can be pruned.

9.2.22 Ability to Verify any specified Job

You now have the ability to tell Bacula which Job should verify instead of automatically verify just the last one.

This feature can be used with VolumeToCatalog, DiskToCatalog and Catalog level.

To verify a given job, just specify the Job jobid in argument when starting the job.

```
*run job=VerifyVolume jobid=1 level=VolumeToCatalog
Run Verify job
JobName:     VerifyVolume
Level:       VolumeToCatalog
Client:      127.0.0.1-fd
FileSet:     Full Set
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  VerifyVol.2010-09-08_14.17.17_03
Verify List: /tmp/regress/working/VerifyVol.bsr
When:        2010-09-08 14:17:31
Priority:    10
OK to run? (yes/mod/no):
```

Chapter 10

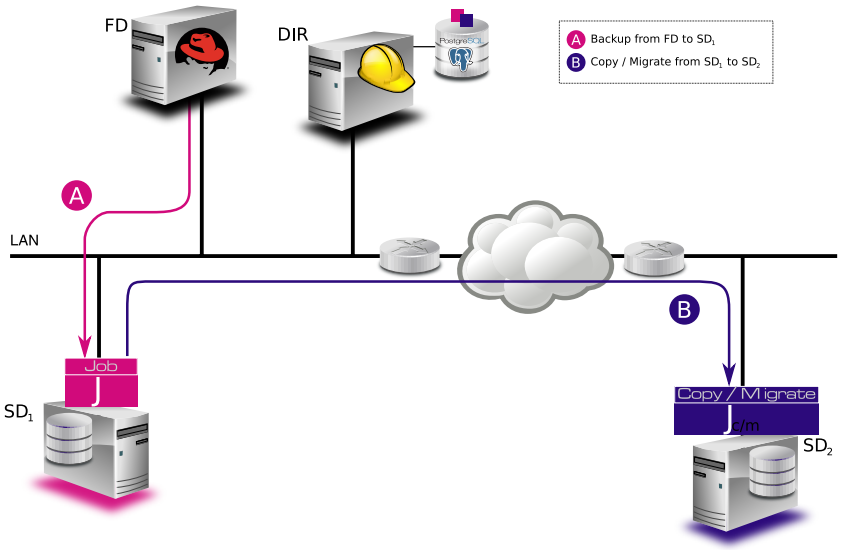
New Features in 7.0.0

This chapter presents the new features that have been added to the various versions of Bacula.

10.1 New Features in 7.0.0

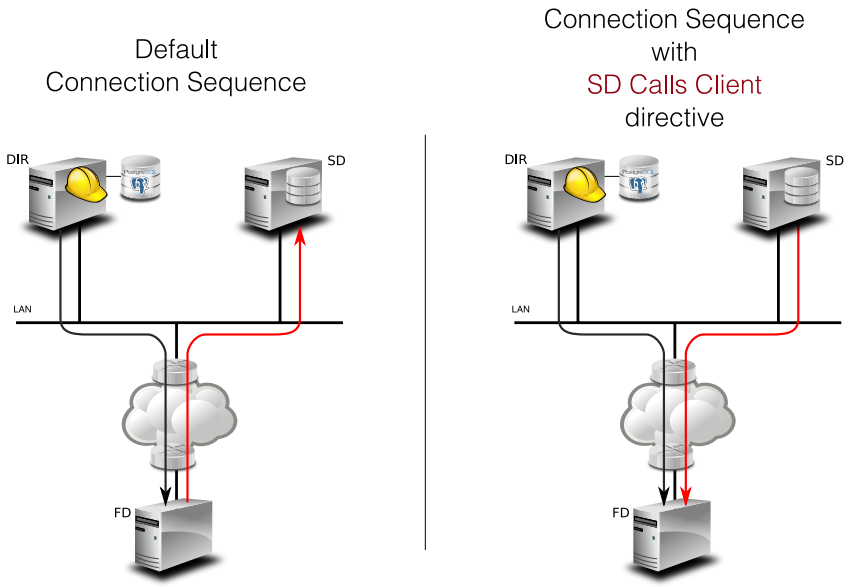
10.1.1 Storage daemon to Storage daemon

Bacula version 7.0 permits SD to SD transfer of Copy and Migration Jobs. This permits what is commonly referred to as replication or off-site transfer of Bacula backups. It occurs automatically, if the source SD and destination SD of a Copy or Migration job are different. The following picture shows how this works.



10.1.2 SD Calls Client

If the **SD Calls Client** directive is set to true in a Client resource any Backup, Restore, Verify, Copy, or Migration Job where the client is involved, the client will wait for the Storage daemon to contact it. By default this directive is set to false, and the Client will call the Storage daemon. This directive can be useful if your Storage daemon is behind a firewall that permits outgoing connections but not incoming one. The following picture shows the communications connection paths in both cases.



10.1.3 Next Pool

In previous versions of Bacula the Next Pool directive could be specified in the Pool resource for use with Migration and Copy Jobs. The Next Pool concept has been extended in Bacula version 7.0.0 to allow you to specify the Next Pool directive in the Job resource as well. If specified in the Job resource, it will override any value specified in the Pool resource.

In addition to being permitted in the Job resource, the **nextpool=xxx** specification can be specified as a run override in the **run** directive of a Schedule resource. Any **nextpool** specification in a **run** directive will override any other specification in either the Job or the Pool.

In general, more information is displayed in the Job log on exactly which Next Pool specification is ultimately used.

10.1.4 status storage

The bconsole **status storage** has been modified to attempt to eliminate duplicate storage resources and only show one that references any given storage daemon. This might be confusing at first, but tends to make a much more compact list of storage resource from which to select if there are multiple storage devices in the same storage daemon.

If you want the old behavior (always display all storage resources) simply add the keyword **select** to the command – i.e. use **status select storage**.

10.1.5 status schedule

A new status command option called **scheduled** has been implemented in bconsole. By default it will display 20 lines of the next scheduled jobs. For example, with the default bacula-dir.conf configuration file, a bconsole command **status scheduled** produces:

```
Scheduled Jobs:
Level        Type    Pri  Scheduled        Job Name     Schedule
=============================================================================
Differential Backup 10  Sun 30-Mar 23:05 BackupClient1 WeeklyCycle
Incremental  Backup 10  Mon 24-Mar 23:05 BackupClient1 WeeklyCycle
```

```
Incremental  Backup 10  Tue 25-Mar 23:05 BackupClient1 WeeklyCycle
...
Full         Backup 11  Mon 24-Mar 23:10 BackupCatalog WeeklyCycleAfterBackup
Full         Backup 11  Wed 26-Mar 23:10 BackupCatalog WeeklyCycleAfterBackup
...
====
```

Note, the output is listed by the Jobs found, and is not sorted chronologically.

This command has a number of options, most of which act as filters:

- **days=nn** This specifies the number of days to list. The default is 10 but can be set from 0 to 500.
- **limit=nn** This specifies the limit to the number of lines to print. The default is 100 but can be any number in the range 0 to 2000.
- **time="YYYY-MM-DD HH:MM:SS"** Sets the start time for listing the scheduled jobs. The default is to use the current time. Note, the time value must be specified inside double quotes and must be in the exact form shown above.
- **schedule=schedule-name** This option restricts the output to the named schedule.
- **job=job-name** This option restricts the output to the specified Job name.

10.1.6 Data Encryption Cipher Configuration

Bacula version 7.0 and later now allows to configure the data encryption cipher and the digest algorithm. The cipher was forced to AES 128, and it is now possible to choose between the following ciphers:

- AES128 (default)
- AES192
- AES256
- blowfish

The digest algorithm was set to SHA1 or SHA256 depending on the local OpenSSL options. We advise you to not modify the PkiDigest default setting. Please, refer to OpenSSL documentation to know about pro and cons on these options.

```
FileDaemon {
  ...
  PkiCipher = AES256
}
```

10.1.7 New Truncate Command

We have added a new truncate command to bconsole, which will truncate a Volume if the Volume is purged and if the Volume is also marked **Action On Purge = Truncate**. This feature was originally added in Bacula version 5.0.1, but the mechanism for actually doing the truncate required the user to enter a command such as:

```
purge volume action=truncate storage=File pool=Default
```

The above command is now simplified to be:

```
truncate storage=File pool=Default
```

10.1.8 Migration/Copy/VirtualFull Performance Enhancements

The Bacula Storage daemon now permits multiple jobs to simultaneously read the same disk Volume, which gives substantial performance enhancements when running Migration, Copy, or VirtualFull jobs that read disk Volumes. Our testing shows that when running multiple simultaneous jobs, the jobs can finish up to ten times faster with this version of Bacula. This is built-in to the Storage daemon, so it happens automatically and transparently.

10.1.9 VirtualFull Backup Consolidation Enhancements

By default Bacula selects jobs automatically for a VirtualFull, however, you may want to create the Virtual backup based on a particular backup (point in time) that exists.

For example, if you have the following backup Jobs in your catalog:

```
+-------+---------+-------+----------+----------+-----------+
| JobId | Name    | Level | JobFiles | JobBytes | JobStatus |
+-------+---------+-------+----------+----------+-----------+
| 1     | Vbackup | F     | 1754     | 50118554 | T         |
| 2     | Vbackup | I     | 1        | 4        | T         |
| 3     | Vbackup | I     | 1        | 4        | T         |
| 4     | Vbackup | D     | 2        | 8        | T         |
| 5     | Vbackup | I     | 1        | 6        | T         |
| 6     | Vbackup | I     | 10       | 60       | T         |
| 7     | Vbackup | I     | 11       | 65       | T         |
| 8     | Save    | F     | 1758     | 50118564 | T         |
+-------+---------+-------+----------+----------+-----------+
```

and you want to consolidate only the first 3 jobs and create a virtual backup equivalent to Job 1 + Job 2 + Job 3, you will use `jobid=3` in the `run` command, then Bacula will select the previous Full backup, the previous Differential (if any) and all subsequent Incremental jobs.

```
run job=Vbackup jobid=3 level=VirtualFull
```

If you want to consolidate a specific job list, you must specify the exact list of jobs to merge in the run command line. For example, to consolidate the last Differential and all subsequent Incremental, you will use `jobid=4,5,6,7` or `jobid=4-7` on the run command line. As one of the Job in the list is a Differential backup, Bacula will set the new job level to Differential. If the list is composed only with Incremental jobs, the new job will have a level set to Incremental.

```
run job=Vbackup jobid=4-7 level=VirtualFull
```

When using this feature, Bacula will automatically discard jobs that are not related to the current Job. For example, specifying `jobid=7,8`, Bacula will discard JobId 8 because it is not part of the same backup Job.

We do not recommend it, but really want to consolidate jobs that have different names (so probably different clients, filesets, etc...), you must use `alljobid=` keyword instead of `jobid=`.

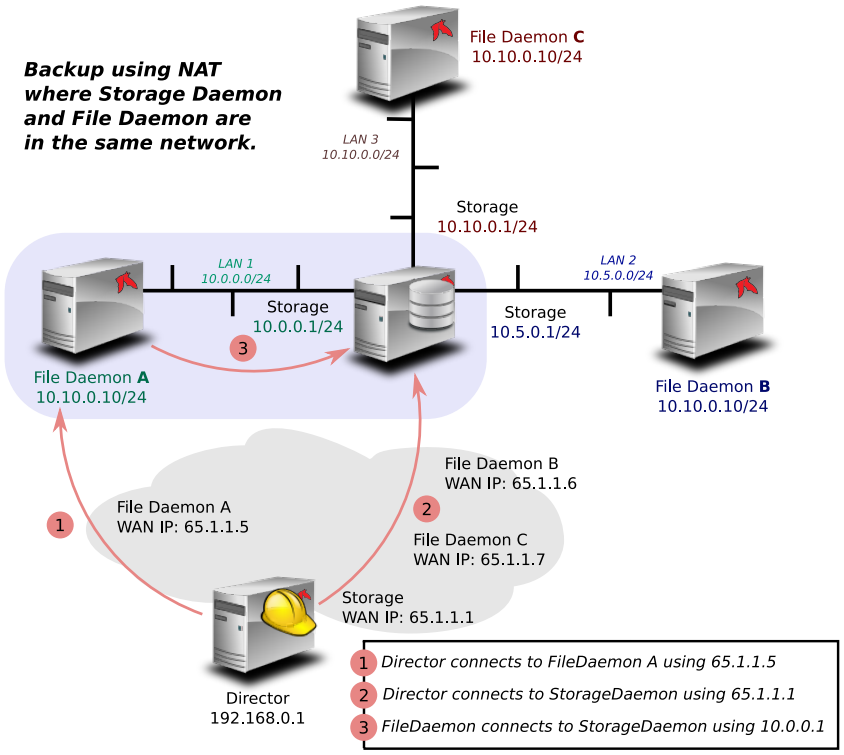
```
run job=Vbackup alljobid=1-3,6-8 level=VirtualFull
```

10.1.10 FD Storage Address

When the Director is behind a NAT, in a WAN area, to connect to the StorageDaemon, the Director uses an "external" ip address, and the FileDaemon should use an "internal" IP address

to contact the StorageDaemon.

The normal way to handle this situation is to use a canonical name such as "storage-server" that will be resolved on the Director side as the WAN address and on the Client side as the LAN address. This is now possible to configure this parameter using the new directive FDStorageAddress in the Storage or Client resource.




```
Storage {
     Name = storage1
     Address = 65.1.1.1
     FD Storage Address = 10.0.0.1
     SD Port = 9103
     ...
}
```

```
 Client {
      Name = client1
      Address = 65.1.1.2
      FD Storage Address = 10.0.0.1
      FD Port = 9102
      ...
 }
```

Note that using the Client FDStorageAddress directive will not allow to use multiple Storage Daemon, all Backup or Restore requests will be sent to the specified FDStorageAddress.

10.1.11 Job Bandwidth Limitation

The new **Job Bandwidth Limitation** directive may be added to the File daemon's and/or Director's configuration to limit the bandwidth used by a Job on a Client. It can be set in the File daemon's conf file for all Jobs run in that File daemon, or it can be set for each Job in the Director's conf file. The speed is always specified in bytes per second.

For example:

```
FileDaemon {
  Name = localhost-fd
  Working Directory = /some/path
  Pid Directory = /some/path
  ...
  Maximum Bandwidth Per Job = 5Mb/s
}
```

The above example would cause any jobs running with the FileDaemon to not exceed 5 megabytes per second of throughput when sending data to the Storage Daemon. Note, the speed is always specified in bytes per second (not in bits per second), and the case (upper/lower) of the specification characters is ignored (i.e. 1MB/s = 1Mb/s).

You may specify the following speed parameter modifiers: k/s (1,000 bytes per second), kb/s (1,024 bytes per second), m/s (1,000,000 bytes per second), or mb/s (1,048,576 bytes per second).

For example:

```
Job {
  Name = locahost-data
  FileSet = FS_localhost
  Accurate = yes
  ...
  Maximum Bandwidth = 5Mb/s
  ...
}
```

The above example would cause Job `localhost-data` to not exceed 5MB/s of throughput when sending data from the File daemon to the Storage daemon.

A new console command `setbandwidth` permits to set dynamically the maximum throughput of a running Job or for future jobs of a Client.

```
* setbandwidth limit=1000 jobid=10
```

Please note that the value specified for the `limit` command line parameter is always in units of 1024 bytes (i.e. the number is multiplied by 1024 to give the number of bytes per second). As a consequence, the above limit of 1000 will be interpreted as a limit of 1000 * 1024 = 1,024,000 bytes per second.

This project was funded by Bacula Systems.

10.1.12 Maximum Concurrent Read Jobs

This is a new directive that can be used in the **bacula-dir.conf** file in the Storage resource. The main purpose is to limit the number of concurrent Copy, Migration, and VirtualFull jobs so that they don't monopolize all the Storage drives causing a deadlock situation where all the drives are allocated for reading but none remain for writing. This deadlock situation can occur when running multiple simultaneous Copy, Migration, and VirtualFull jobs.

The default value is set to 0 (zero), which means there is no limit on the number of read jobs. Note, limiting the read jobs does not apply to Restore jobs, which are normally started by hand.

A reasonable value for this directive is one half the number of drives that the Storage resource has rounded down. Doing so, will leave the same number of drives for writing and will generally avoid over committing drives and a deadlock.

10.1.13 Director job Codes in Message Resource Commands

Before submitting the specified mail command to the operating system, Bacula performs character substitution like in Runscript commands. Bacula will now perform also specific Director character substitution.

The code for this feature was contributed by Bastian Friedrich.

10.1.14 Additions to RunScript variables

The following variables are now available in runscripts:

- current PID using %P
- if the job is a clone job using %C

```
RunAfterJob = "/bin/echo Pid=%P isCloned=%C"
```

10.1.15 Read Only Storage Devices

This version of Bacula permits defining a Storage daemon device to be read-only. That is if the **ReadOnly** directive is specified and enabled, the drive can only be used for read operations. The the **ReadOnly** directive can be defined in any bacula-sd.conf Device resource, and is most useful to reserve one or more drives for restores. An example is:

```
Read Only = yes
```

10.1.16 New Prune "Expired" Volume Command

It is now possible to prune all volumes (from a pool, or globally) that are "expired". This option can be scheduled after or before the backup of the Catalog and can be combined with the Truncate On Purge option. The Expired Prune option can be used instead of the `manual_prune.pl` script.

```
* prune expired volumes

* prune expired volumes pool=FullPool
```

To schedule this option automatically, it can be added to the BackupCatalog job definition.

```
Job {
  Name = CatalogBackup
  ...
  RunScript {
    Console = "prune expired volume yes"
    RunsWhen = Before
  }
}
```

10.1.17 Hardlink Performance Enhancements

If you use a program such as Cyrus IMAP that creates very large numbers of hardlinks, the time to build the interactive restore tree can be excessively long. This version of Bacula has a new feature that automatically keeps the hardlinks associated with the restore tree in memory, which consumes a bit more memory but vastly speeds up building the tree. If the memory usage is too big for your system, you can reduce the amount of memory used during the restore command by adding the option **optimizespeed=false** on the bconsole run command line.

This feature was developed by Josip Almasi, and enhanced to be runtime dynamic by Kern Sibbald.

10.1.18 DisableCommand Directive

There is a new Directive named **Disable Command** that can be put in the File daemon Client or Director resource. If it is in the Client, it applies globally, otherwise the directive applies only to the Director in which it is found. The Disable Command adds security to your File daemon by disabling certain commands. The commands that can be disabled are:

```
backup
cancel
setdebug=
setbandwidth=
estimate
fileset
JobId=
level =
restore
endrestore
session
status
.status
storage
verify
RunBeforeNow
RunBeforeJob
RunAfterJob
Run
accurate
```

On or more of these command keywords can be placed in quotes and separated by spaces on the Disable Command directive line. Note: the commands must be written exactly as they appear above.

10.1.19 Multiple Console Directors

Support for multiple bconsole and bat Directors in the bconsole.conf and bat.conf files has been implemented and/or improved.

10.1.20 Restricted Consoles

Better support for Restricted consoles has been implement for bconsole and bat.

10.1.21 Configuration Files

In previous versions of Bacula the configuration files for each component were limited to a maximum of 499 bytes per configuration file line. This version of Bacula permits unlimited input line lengths. This can be especially useful for specifying more complicated Migration/Copy SQL statements and in creating long restricted console ACL lists.

10.1.22 Maximum Spawned Jobs

The Job resource now permits specifying a number of **Maximum Spawned Jobs**. The default is 600. This directive can be useful if you have big hardware and you do a lot of Migration/Copy jobs which start at the same time. In prior versions of Bacula, Migration/Copy was limited to spawning a maximum of 100 jobs at a time.

10.1.23 Progress Meter

The new File daemon has been enhanced to send its progress (files processed and bytes written) to the Director every 30 seconds. These figures can then be displayed with a bconsole **status dir** command.

10.1.24 Scheduling a 6th Week

Prior version of Bacula permits specifying 1st through 5th week of a month (first through fifth) as a keyword on the **run** directive of a Schedule resource. This version of Bacula also permits specifying the 6th week of a month with the keyword **sixth** or **6th**.

10.1.25 Scheduling the Last Day of a Month

This version of Bacula now permits specifying the **lastday** keyword in the **run** directive of a Schedule resource. If **lastday** is specified, it will apply only to those months specified on the **run** directive. Note: by default all months are specified.

10.1.26 Improvements to Cancel and Restart bconsole Commands

The Restart bconsole command now allow selection of either canceled or failed jobs to be restarted. In addition both the **cancel** and **restart** bconsole commands permit entering a number of JobIds separated by commas or a range of JobIds indicated by a dash between the begin and end range (e.g. 3-10). Finally the two commands also allow one to enter the special keyword **all** to select all the appropriate Jobs.

10.1.27 bconsole Performance Improvements

In previous versions of Bacula certain bconsole commands could wait a long time due to catalog lock contention. This was especially noticeable when a large number of jobs were running and putting their attributes into the catalog. This version uses a separate catalog connection that should significantly enhance performance.

10.1.28 New .bvfs_decode_lstat Command

There is a new bconsole command, which is **.bvfs_decode_lstat** it requires one argument, which is **lstat="lstat value to decode"**. An example command in bconsole and the output might be:

```
.bvfs_decode_lstat lstat="A A EHt B A A A JP BAA B BTL/A7 BTL/A7 BTL/A7 A A C"

st_nlink=1
st_mode=16877
st_uid=0
st_gid=0
st_size=591
st_blocks=1
st_ino=0
st_ctime=1395650619
st_mtime=1395650619
st_mtime=1395650619
st_dev=0
LinkFI=0
```

New Debug Options

In Bacula Enterprise version 8.0 and later, we introduced new options to the `setdebug` command.

If the `options` parameter is set, the following arguments can be used to control debug functions.

- 0 clear debug flags
- i Turn off, ignore bwrite() errors on restore on File Daemon
- d Turn off decomp of BackupRead() streams on File Daemon
- t Turn on timestamp in traces
- T Turn off timestamp in traces
- c Truncate trace file if trace file is activated
- l Turn on recoding events on P() and V()
- p Turn on the display of the event ring when doing a bactrace

The following command will truncate the trace file and will turn on timestamps in the trace file.

```
* setdebug level=10 trace=1 options=ct fd
```

It is now possible to use *class* of debug messages called `tags` to control the debug output of Bacula daemons.

- all Display all debug messages
- bvfs Display BVFS debug messages
- sql Display SQL related debug messages
- memory Display memory and poolmem allocation messages
- scheduler Display scheduler related debug messages

```
* setdebug level=10 tags=bvfs,sql,memory
* setdebug level=10 tags=!bvfs

# bacula-dir -t -d 200,bvfs,sql
```

The tags option is composed of a list of tags, tags are separated by "," or "+" or "-" or "!". To disable a specific tag, use "-" or "!" in front of the tag. Note that more tags will come in future versions.

Chapter 11

New Features in 5.2.13

This chapter presents the new features that have been added to the current Community version of Bacula that is now released.

11.0.1 Additions to RunScript variables

You can have access to Director name using %D in your runscript command.

```
RunAfterJob = "/bin/echo Director=%D
```

11.1 New Features in 5.2.1

This chapter presents the new features were added in the Community release version 5.2.1.

There are additional features (plugins) available in the Enterprise version that are described in another chapter. A subscription to Bacula Systems is required for the Enterprise version.

11.1.1 LZO Compression

LZO compression has been to the File daemon. From the user's point of view, it works like the GZIP compression (just replace **compression=GZIP** with **compression=LZO**).

For example:

```
Include {
   Options {compression=LZO }
   File = /home
   File = /data
}
```

LZO provides a much faster compression and decompression speed but lower compression ratio than GZIP. It is a good option when you backup to disk. For tape, the hardware compression is almost always a better option.

LZO is a good alternative for GZIP1 when you don't want to slow down your backup. With a modern CPU it should be able to run almost as fast as:

- your client can read data from disk. Unless you have very fast disks like SSD or large/fast RAID array.
- the data transfers between the file daemon and the storage daemon even on a 1Gb/s link.

Note, Bacula uses compression level LZO1X-1.

The code for this feature was contributed by Laurent Papier.

11.1.2 New Tray Monitor

Since the old integrated Windows tray monitor doesn't work with recent Windows versions, we have written a new Qt Tray Monitor that is available for both Linux and Windows. In addition to all the previous features, this new version allows you to run Backups from the tray monitor menu.

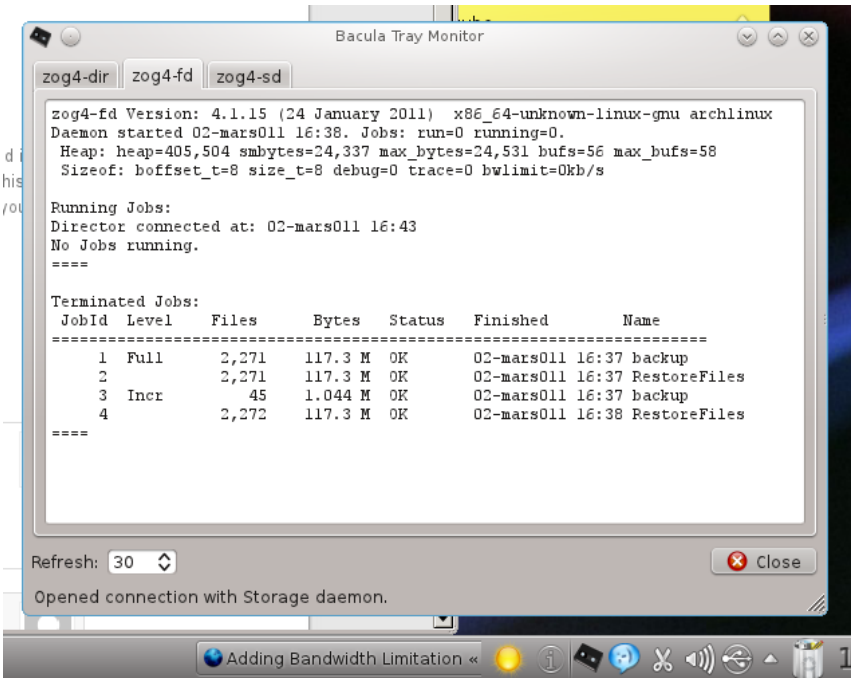


Figure 11.1: New tray monitor

To be able to run a job from the tray monitor, you need to allow specific commands in the Director monitor console:

```
Console {
    Name = win2003-mon
    Password = "xxx"
    CommandACL = status, .clients, .jobs, .pools, .storage, .filesets, .messages, run
    ClientACL = *all*                # you can restrict to a specific host
    CatalogACL = *all*
    JobACL = *all*
    StorageACL = *all*
    ScheduleACL = *all*
    PoolACL = *all*
    FileSetACL = *all*
    WhereACL = *all*
}
```

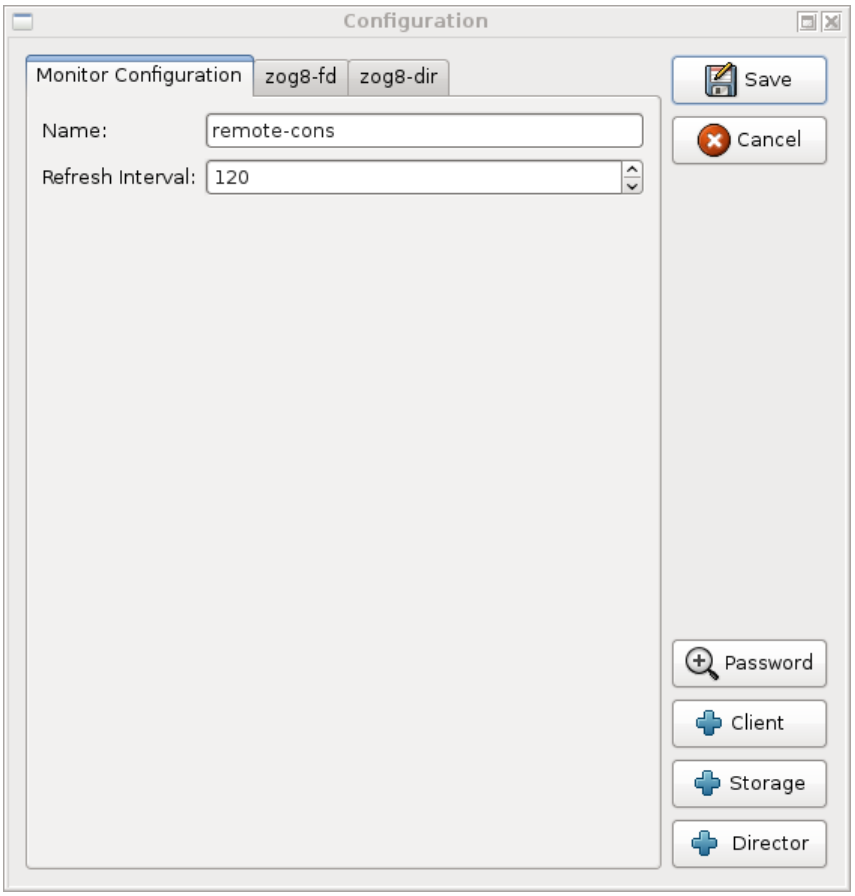


Figure 11.2: Run a Job through the new tray monitor

This project was funded by Bacula Systems and is available with Bacula the Enterprise Edition and the Community Edition.

11.1.3 Purge Migration Job

The new **Purge Migration Job** directive may be added to the Migration Job definition in the Director's configuration file. When it is enabled the Job that was migrated during a migration will be purged at the end of the migration job.

For example:

```
Job {
  Name = "migrate-job"
  Type = Migrate
  Level = Full
  Client = localhost-fd
  FileSet = "Full Set"
  Messages = Standard
  Storage = DiskChanger
  Pool = Default
  Selection Type = Job
  Selection Pattern = ".*Save"
...
  Purge Migration Job = yes
}
```

This project was submitted by Dunlap Blake; testing and documentation was funded by Bacula Systems.

11.1.4 Changes in Bvfs (Bacula Virtual FileSystem)

Bat has now a bRestore panel that uses Bvfs to display files and directories.

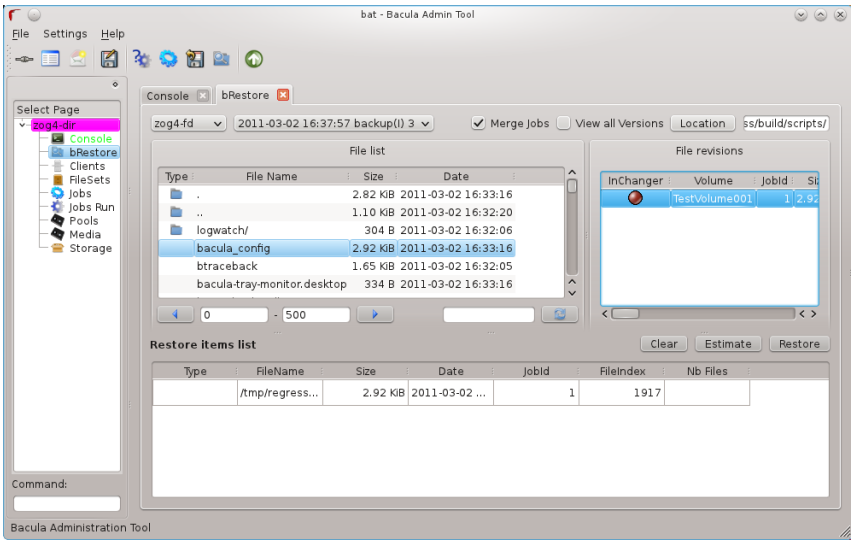


Figure 11.3: Bat Brestore Panel

the Bvfs module works correctly with BaseJobs, Copy and Migration jobs.

This project was funded by Bacula Systems.

General notes

- All fields are separated by a tab
- You can specify `limit=` and `offset=` to list smoothly records in very big directories
- All operations (except cache creation) are designed to run instantly
- At this time, Bvfs works faster on PostgreSQL than MySQL catalog. If you can contribute new faster SQL queries we will be happy, else don't complain about speed.
- The cache creation is dependent of the number of directories. As Bvfs shares information across jobs, the first creation can be slow
- All fields are separated by a tab
- Due to potential encoding problem, it's advised to always use pathid in queries.

Get dependent jobs from a given JobId

Bvfs allows you to query the catalog against any combination of jobs. You can combine all Jobs and all FileSet for a Client in a single session.

To get all JobId needed to restore a particular job, you can use the `.bvfs_get_jobids` command.

```
.bvfs_get_jobids jobid=num [all]
```

```
.bvfs_get_jobids jobid=10
1,2,5,10
.bvfs_get_jobids jobid=10 all
1,2,3,5,10
```

In this example, a normal restore will need to use JobIds 1,2,5,10 to compute a complete restore of the system.

With the `all` option, the Director will use all defined FileSet for this client.

Generating Bvfs cache

The `.bvfs_update` command computes the directory cache for jobs specified in argument, or for all jobs if unspecified.

```
.bvfs_update [jobid=numlist]
```

Example:

```
.bvfs_update jobid=1,2,3
```

You can run the cache update process in a RunScript after the catalog backup.

Get all versions of a specific file

Bvfs allows you to find all versions of a specific file for a given Client with the `.bvfs_version` command. To avoid problems with encoding, this function uses only PathId and FilenameId. The jobid argument is mandatory but unused.

```
.bvfs_versions client=filedaemon pathid=num filenameid=num jobid=1
PathId FilenameId FileId JobId LStat Md5 VolName Inchanger
PathId FilenameId FileId JobId LStat Md5 VolName Inchanger
...
```

Example:

```
.bvfs_versions client=localhost-fd pathid=1 fnid=47 jobid=1
1  47  52  12  gD HRid IGk D Po Po A P BAA I A   /uPgWaxMgKZlnMti7LChyA  Vol1  1
```

List directories

Bvfs allows you to list directories in a specific path.

```
.bvfs_lsdirs pathid=num path=/apath jobid=numlist limit=num offset=num
PathId  FilenameId  FileId  JobId  LStat  Path
PathId  FilenameId  FileId  JobId  LStat  Path
PathId  FilenameId  FileId  JobId  LStat  Path
...
```

You need to pathid or path. Using path="" will list "/" on Unix and all drives on Windows. If FilenameId is 0, the record listed is a directory.

```
.bvfs_lsdirs pathid=4 jobid=1,11,12
4       0       0       0       A A A A A A A A A A A A A A     .
5       0       0       0       A A A A A A A A A A A A A A     ..
3       0       0       0       A A A A A A A A A A A A A A     regress/
```

In this example, to list directories present in regress/, you can use

```
.bvfs_lsdirs pathid=3 jobid=1,11,12
3       0       0       0       A A A A A A A A A A A A A A     .
4       0       0       0       A A A A A A A A A A A A A A     ..
2       0       0       0       A A A A A A A A A A A A A A     tmp/
```

List files

Bvfs allows you to list files in a specific path.

```
.bvfs_lsfiles pathid=num path=/apath jobid=numlist limit=num offset=num
PathId  FilenameId  FileId  JobId  LStat  Path
PathId  FilenameId  FileId  JobId  LStat  Path
PathId  FilenameId  FileId  JobId  LStat  Path
...
```

You need to pathid or path. Using path="" will list "/" on Unix and all drives on Windows. If FilenameId is 0, the record listed is a directory.

```
.bvfs_lsfiles pathid=4 jobid=1,11,12
4       0       0       0       A A A A A A A A A A A A A A     .
5       0       0       0       A A A A A A A A A A A A A A     ..
1       0       0       0       A A A A A A A A A A A A A A     regress/
```

In this example, to list files present in `regress/`, you can use

```
.bvfs_lsfiles pathid=1 jobid=1,11,12
1   47   52   12    gD HRid IGk BAA I BMqcPH BMqcPE BMqe+t A     titi
1   49   53   12    gD HRid IGk BAA I BMqe/K BMqcPE BMqe+t B     toto
1   48   54   12    gD HRie IGk BAA I BMqcPH BMqcPE BMqe+3 A     tutu
1   45   55   12    gD HRid IGk BAA I BMqe/K BMqcPE BMqe+t B     ficheriro1.txt
1   46   56   12    gD HRie IGk BAA I BMqe/K BMqcPE BMqe+3 D     ficheriro2.txt
```

Restore set of files

Bvfs allows you to create a SQL table that contains files that you want to restore. This table can be provided to a restore command with the file option.

```
.bvfs_restore fileid=numlist dirid=numlist hardlink=numlist path=b2num
OK
restore file=?b2num ...
```

To include a directory (with `dirid`), Bvfs needs to run a query to select all files. This query could be time consuming.

`hardlink` list is always composed of a series of two numbers (jobid, fileindex). This information can be found in the LinkFI field of the LStat packet.

The `path` argument represents the name of the table that Bvfs will store results. The format of this table is `b2[0-9]+`. (Should start by b2 and followed by digits).

Example:

```
.bvfs_restore fileid=1,2,3,4 hardlink=10,15,10,20 jobid=10 path=b20001
OK
```

Cleanup after Restore

To drop the table used by the restore command, you can use the `.bvfs_cleanup` command.

```
.bvfs_cleanup path=b20001
```

Clearing the BVFS Cache

To clear the BVFS cache, you can use the `.bvfs_clear_cache` command.

```
.bvfs_clear_cache yes
OK
```

11.1.5 Changes in the Pruning Algorithm

We rewrote the job pruning algorithm in this version. Previously, in some users reported that the pruning process at the end of jobs was very long. It should not be longer the case. Now, Bacula won't prune automatically a Job if this particular Job is needed to restore data. Example:

```
JobId: 1  Level: Full
JobId: 2  Level: Incremental
JobId: 3  Level: Incremental
JobId: 4  Level: Differential
.. Other incrementals up to now
```

In this example, if the Job Retention defined in the Pool or in the Client resource causes that Jobs with Jobid in 1,2,3,4 can be pruned, Bacula will detect that JobId 1 and 4 are essential to restore data at the current state and will prune only JobId 2 and 3.

`Important`, this change affect only the automatic pruning step after a Job and the `prune jobs` Bconsole command. If a volume expires after the `VolumeRetention` period, important jobs can be pruned.

11.1.6 Ability to Verify any specified Job

You now have the ability to tell Bacula which Job should verify instead of automatically verify just the last one.

This feature can be used with VolumeToCatalog, DiskToCatalog and Catalog level.

To verify a given job, just specify the Job jobid in argument when starting the job.

```
*run job=VerifyVolume jobid=1 level=VolumeToCatalog
Run Verify job
JobName:     VerifyVolume
Level:       VolumeToCatalog
Client:      127.0.0.1-fd
FileSet:     Full Set
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  VerifyVol.2010-09-08_14.17.17_03
Verify List: /tmp/regress/working/VerifyVol.bsr
When:        2010-09-08 14:17:31
Priority:    10
OK to run? (yes/mod/no):
```

This project was funded by Bacula Systems and is available with Bacula Enterprise Edition and Community Edition.

11.1.7 Additions to RunScript variables

You can have access to JobBytes and JobFiles using %b and %F in your runscript command. The Client address is now available through %h.

```
RunAfterJob = "/bin/echo Job=%j JobBytes=%b JobFiles=%F ClientAddress=%h"
```

11.1.8 Additions to the Plugin API

The bfuncs structure has been extended to include a number of new entrypoints.

bfuncs

The bFuncs structure defines the callback entry points within Bacula that the plugin can use register events, get Bacula values, set Bacula values, and send messages to the Job output or debug output.

The exact definition as of this writing is:

```
typedef struct s_baculaFuncs {
   uint32_t size;
   uint32_t version;
   bRC (*registerBaculaEvents)(bpContext *ctx, ...);
   bRC (*getBaculaValue)(bpContext *ctx, bVariable var, void *value);
   bRC (*setBaculaValue)(bpContext *ctx, bVariable var, void *value);
   bRC (*JobMessage)(bpContext *ctx, const char *file, int line,
       int type, utime_t mtime, const char *fmt, ...);
   bRC (*DebugMessage)(bpContext *ctx, const char *file, int line,
       int level, const char *fmt, ...);
   void *(*baculaMalloc)(bpContext *ctx, const char *file, int line,
       size_t size);
   void (*baculaFree)(bpContext *ctx, const char *file, int line, void *mem);

   /* New functions follow */
   bRC (*AddExclude)(bpContext *ctx, const char *file);
   bRC (*AddInclude)(bpContext *ctx, const char *file);
   bRC (*AddIncludeOptions)(bpContext *ctx, const char *opts);
   bRC (*AddRegex)(bpContext *ctx, const char *item, int type);
   bRC (*AddWild)(bpContext *ctx, const char *item, int type);
   bRC (*checkChanges)(bpContext *ctx, struct save_pkt *sp);

} bFuncs;
```

AddExclude can be called to exclude a file. The file string passed may include wildcards that will be interpreted by the **fnmatch** subroutine. This function can be called multiple times, and each time the file specified will be added to the list of files to be excluded. Note, this function only permits adding excludes of specific file or directory names, or files matched by the rather simple fnmatch mechanism. See below for information on doing wild-card and regex excludes.

NewPreInclude can be called to create a new Include block. This block will be added after the current defined Include block. This function can be called multiple times, but each time, it will create a new Include section (not normally needed). This function should be called only if you want to add an entirely new Include block.

NewInclude can be called to create a new Include block. This block will be added before any user defined Include blocks. This function can be called multiple times, but each time, it will create a new Include section (not normally needed). This function should be called only if you want to add an entirely new Include block.

AddInclude can be called to add new files/directories to be included. They are added to the current Include block. If NewInclude has not been included, the current Include block is the last one that the user created. This function should be used only if you want to add totally new files/directories to be included in the backup.

NewOptions adds a new Options block to the current Include in front of any other Options blocks. This permits the plugin to add exclude directives (wild-cards and regexes) in front of the user Options, and thus prevent certain files from being backed up. This can be useful if the plugin backs up files, and they should not be also backed up by the main Bacula code. This function may be called multiple times, and each time, it creates a new prepended Options block. Note: normally you want to call this entry point prior to calling AddOptions, AddRegex, or AddWild.

AddOptions allows the plugin it set options in the current Options block, which is normally created with the NewOptions call just prior to adding Include Options. The permitted options are passed as a character string, where each character has a specific meaning as defined below:

a always replace files (default).

e exclude rather than include.

h no recursion into subdirectories.

H do not handle hard links.

i ignore case in wildcard and regex matches.

M compute an MD5 sum.

p use a portable data format on Windows (not recommended).

R backup resource forks and Findr Info.

r read from a fifo

S1 compute an SHA1 sum.

S2 compute an SHA256 sum.

S3 comput an SHA512 sum.

s handle sparse files.

m use st_mtime only for file differences.

k restore the st_atime after accessing a file.

A enable ACL backup.

Vxxx: specify verify options. Must terminate with :

Cxxx: specify accurate options. Must terminate with :

Jxxx: specify base job Options. Must terminate with :

Pnnn: specify integer nnn paths to strip. Must terminate with :

w if newer

Zn specify gzip compression level n.

K do not use st_atime in backup decision.

c check if file changed during backup.

N honor no dump flag.

X enable backup of extended attributes.

AddRegex adds a regex expression to the current Options block. The following options are permitted:

(a blank) regex applies to whole path and filename.

F regex applies only to the filename (directory or path stripped).

D regex applies only to the directory (path) part of the name.

AddWild adds a wildcard expression to the current Options block. The following options are permitted:

(a blank) regex applies to whole path and filename.

F regex applies only to the filename (directory or path stripped).

D regex applies only to the directory (path) part of the name.

checkChanges call the `check_changes()` function in Bacula code that can use Accurate code to compare the file information in argument with the previous file information. The `delta_seq` attribute of the `save_pkt` will be updated, and the call will return `bRC_Seen` if the core code wouldn't decide to backup it.

Bacula events

The list of events has been extended to include:

```
typedef enum {
  bEventJobStart        = 1,
  bEventJobEnd          = 2,
  bEventStartBackupJob  = 3,
  bEventEndBackupJob    = 4,
  bEventStartRestoreJob = 5,
  bEventEndRestoreJob   = 6,
  bEventStartVerifyJob  = 7,
  bEventEndVerifyJob    = 8,
  bEventBackupCommand   = 9,
  bEventRestoreCommand  = 10,
  bEventLevel           = 11,
  bEventSince           = 12,

  /* New events */
  bEventCancelCommand                   = 13,
  bEventVssBackupAddComponents          = 14,
  bEventVssRestoreLoadComponentMetadata = 15,
  bEventVssRestoreSetComponentsSelected = 16,
  bEventRestoreObject                   = 17,
  bEventEndFileSet                      = 18,
  bEventPluginCommand                   = 19,
  bEventVssBeforeCloseRestore           = 20,
  bEventVssPrepareSnapshot              = 21

} bEventType;
```

bEventCancelCommand is called whenever the currently running Job is canceled */

bEventVssBackupAddComponents

bEventVssPrepareSnapshot is called before creating VSS snapshots, it provides a char[27] table where the plugin can add Windows drives that will be used during the Job. You need to add them without duplicates, and you can use in `fd_common.h` `add_drive()` and `copy_drives()` for this purpose.

11.1.9 ACL enhancements

The following enhancements are made to the Bacula Filed with regards to Access Control Lists (ACLs)

- Added support for AIX 5.3 and later new aclx_get interface which supports POSIX and NFSv4 ACLs.
- Added support for new acl types on FreeBSD 8.1 and later which supports POSIX and NFSv4 ACLs.
- Some generic cleanups for internal ACL handling.
- Fix for acl storage on OSX
- Cleanup of configure checks for ACL detection, now configure only tests for a certain interface type based on the operating system this should give less false positives on detection. Also when ACLs are detected no other acl checks are performed anymore.

This project was funded by Planets Communications B.V. and ELM Consultancy B.V. and is available with Bacula Enterprise Edition and Community Edition.

11.1.10 XATTR enhancements

The following enhancements are made to the Bacula Filed with regards to Extended Attributes (XATTRs)

- Added support for IRIX extended attributes using the attr_get interface.
- Added support for Tru64 (OSF1) extended attributes using the getproplist interface.
- Added support for AIX extended attributes available in AIX 6.x and higher using the listea/getea/setea interface.
- Added some debugging to generic xattr code so it easier to debug.
- Cleanup of configure checks for XATTR detection, now configure only tests for a certain interface type based on the operating system this should give less false positives on detection. Also when xattrs are detected no other xattr checks are performed anymore.

This project was funded by Planets Communications B.V. and ELM Consultancy B.V. and is available with Bacula Enterprise Edition and Community Edition.

11.1.11 Class Based Database Backend Drivers

The main Bacula Director code is independent of the SQL backend in version 5.2.0 and greater. This means that the Bacula Director can be packaged by itself, then each of the different SQL backends supported can be packaged separately. It is possible to build all the DB backends at the same time by including multiple database options at the same time.

./configure can be run with multiple database configure options.

```
--with-mysql
--with-postgresql
```

Order of testing for databases is:

- postgresql
- mysql

Each configured backend generates a file named: `libbaccats-<sql_backend_name>-<version>.so`
A dummy catalog library is created named libbaccats-version.so

At configure time the first detected backend is used as the so called default backend and at install time the dummy `libbaccats-<version>.so` is replaced with the default backend type.

If you configure all three backends you get three backend libraries and the postgresql gets installed as the default.

When you want to switch to another database, first save any old catalog you may have then you can copy one of the three backend libraries over the `libbaccats-<version>.so` e.g.

An actual command, depending on your Bacula version might be:

```
cp libbaccats-postgresql-5.2.2.so libbaccats-5.2.2.so
```

where the 5.2.2 must be replaced by the Bacula release version number.

Then you must update the default backend in the following files:

```
create_bacula_database
drop_bacula_database
drop_bacula_tables
grant_bacula_privileges
make_bacula_tables
make_catalog_backup
update_bacula_tables
```

And re-run all the above scripts. Please note, this means you will have a new empty database and if you had a previous one it will be lost.

All current database backend drivers for catalog information are rewritten to use a set of multi inherited C++ classes which abstract the specific database specific internals and make sure we have a more stable generic interface with the rest of SQL code. From now on there is a strict boundary between the SQL code and the low-level database functions. This new interface should also make it easier to add a new backend for a currently unsupported database. An extra bonus of the new code is that you can configure multiple backends in the configure and build all backends in one compile session and select the correct database backend at install time. This should make it a lot easier for packages maintainers.

We also added cursor support for PostgreSQL backend, this improves memory usage for large installation.

This project was implemented by Planets Communications B.V. and ELM Consultancy B.V. and Bacula Systems and is available with both the Bacula Enterprise Edition and the Community Edition.

11.1.12 Hash List Enhancements

The htable hash table class has been extended with extra hash functions for handling next to char pointer hashes also 32 bits and 64 bits hash keys. Also the hash table initialization routines have been enhanced with support for passing a hint as to the number of initial pages to use for the size of the hash table. Until now the hash table always used a fixed value of 10 Mb. The private hash functions of the mountpoint entry cache have been rewritten to use the new htable class with a small memory footprint.

This project was funded by Planets Communications B.V. and ELM Consultancy B.V. and Bacula Systems and is available with Bacula Enterprise Edition and Community Edition.

11.2 Release Version 5.0.3

There are no new features in version 5.0.2. This version simply fixes a number of bugs found in version 5.0.1 during the ongoing development process.

11.3 Release Version 5.0.2

There are no new features in version 5.0.2. This version simply fixes a number of bugs found in version 5.0.1 during the ongoing development process.

11.4 New Features in 5.0.1

This chapter presents the new features that are in the released Bacula version 5.0.1. This version mainly fixes a number of bugs found in version 5.0.0 during the ongoing development process.

11.4.1 Truncate Volume after Purge

The Pool directive **ActionOnPurge=Truncate** instructs Bacula to truncate the volume when it is purged with the new command `purge volume action`. It is useful to prevent disk based volumes from consuming too much space.

```
Pool {
  Name = Default
  Action On Purge = Truncate
  ...
}
```

As usual you can also set this property with the `update volume` command

```
*update volume=xxx ActionOnPurge=Truncate
*update volume=xxx actiononpurge=None
```

To ask Bacula to truncate your `Purged` volumes, you need to use the following command in interactive mode or in a RunScript as shown after:

```
*purge volume action=truncate storage=File allpools
# or by default, action=all
*purge volume action storage=File pool=Default
```

This is possible to specify the volume name, the media type, the pool, the storage, etc... (see `help purge`) Be sure that your storage device is idle when you decide to run this command.

```
Job {
 Name = CatalogBackup
 ...
 RunScript {
   RunsWhen=After
   RunsOnClient=No
   Console = "purge volume action=all allpools storage=File"
 }
}
```

Important note: This feature doesn't work as expected in version 5.0.0. Please do not use it before version 5.0.1.

11.4.2 Allow Higher Duplicates

This directive did not work correctly and has been depreciated (disabled) in version 5.0.1. Please remove it from your bacula-dir.conf file as it will be removed in a future release.

11.4.3 Cancel Lower Level Duplicates

This directive was added in Bacula version 5.0.1. It compares the level of a new backup job to old jobs of the same name, if any, and will kill the job which has a lower level than the other one. If the levels are the same (i.e. both are Full backups), then nothing is done and the other Cancel XXX Duplicate directives will be examined.

11.5 New Features in 5.0.0

11.5.1 Maximum Concurrent Jobs for Devices

Maximum Concurrent Jobs is a new Device directive in the Storage Daemon configuration permits setting the maximum number of Jobs that can run concurrently on a specified Device. Using this directive, it is possible to have different Jobs using multiple drives, because when the Maximum Concurrent Jobs limit is reached, the Storage Daemon will start new Jobs on any other available compatible drive. This facilitates writing to multiple drives with multiple Jobs that all use the same Pool.

This project was funded by Bacula Systems.

11.5.2 Restore from Multiple Storage Daemons

Previously, you were able to restore from multiple devices in a single Storage Daemon. Now, Bacula is able to restore from multiple Storage Daemons. For example, if your full backup runs on a Storage Daemon with an autochanger, and your incremental jobs use another Storage Daemon with lots of disks, Bacula will switch automatically from one Storage Daemon to an other within the same Restore job.

You must upgrade your File Daemon to version 3.1.3 or greater to use this feature.

This project was funded by Bacula Systems with the help of Equiinet.

11.5.3 File Deduplication using Base Jobs

A base job is sort of like a Full save except that you will want the FileSet to contain only files that are unlikely to change in the future (i.e. a snapshot of most of your system after installing it). After the base job has been run, when you are doing a Full save, you specify one or more Base jobs to be used. All files that have been backed up in the Base job/jobs but not modified will then be excluded from the backup. During a restore, the Base jobs will be automatically pulled in where necessary.

This is something none of the competition does, as far as we know (except perhaps BackupPC, which is a Perl program that saves to disk only). It is big win for the user, it makes Bacula stand out as offering a unique optimization that immediately saves time and money. Basically, imagine that you have 100 nearly identical Windows or Linux machine containing the OS and user files. Now for the OS part, a Base job will be backed up once, and rather than making 100 copies of the OS, there will be only one. If one or more of the systems have some files updated, no problem, they will be automatically restored.

See the Base Job Chapter for more information.

This project was funded by Bacula Systems.

11.5.4 AllowCompression = <yes|no>

This new directive may be added to Storage resource within the Director's configuration to allow users to selectively disable the client compression for any job which writes to this storage resource.

For example:

```
Storage {
  Name = UltriumTape
  Address = ultrium-tape
  Password = storage_password # Password for Storage Daemon
  Device = Ultrium
  Media Type = LTO 3
  AllowCompression = No # Tape drive has hardware compression
}
```

The above example would cause any jobs running with the UltriumTape storage resource to run without compression from the client file daemons. This effectively overrides any compression settings defined at the FileSet level.

This feature is probably most useful if you have a tape drive which supports hardware compression. By setting the `AllowCompression = No` directive for your tape drive storage resource, you can avoid additional load on the file daemon and possibly speed up tape backups.

This project was funded by Collaborative Fusion, Inc.

11.5.5 Accurate Fileset Options

In previous versions, the accurate code used the file creation and modification times to determine if a file was modified or not. Now you can specify which attributes to use (time, size, checksum, permission, owner, group, ...), similar to the Verify options.

```
FileSet {
  Name = Full
  Include = {
    Options {
       Accurate = mcs
       Verify   = pin5
    }
    File = /
  }
}
```

i compare the inodes

p compare the permission bits

n compare the number of links

u compare the user id

g compare the group id

s compare the size

a compare the access time

m compare the modification time (st_mtime)

c compare the change time (st_ctime)

d report file size decreases

5 compare the MD5 signature

1 compare the SHA1 signature

Important note: If you decide to use checksum in Accurate jobs, the File Daemon will have to read all files even if they normally would not be saved. This increases the I/O load, but also the accuracy of the deduplication. By default, Bacula will check modification/creation time and size.

This project was funded by Bacula Systems.

11.5.6 Tab-completion for Bconsole

If you build `bconsole` with readline support, you will be able to use the new auto-completion mode. This mode supports all commands, gives help inside command, and lists resources when required. It works also in the restore mode.

To use this feature, you should have readline development package loaded on your system, and use the following option in configure.

```
./configure --with-readline=/usr/include/readline --disable-conio ...
```

The new bconsole won't be able to tab-complete with older directors.

This project was funded by Bacula Systems.

11.5.7 Pool File and Job Retention

We added two new Pool directives, `FileRetention` and `JobRetention`, that take precedence over Client directives of the same name. It allows you to control the Catalog pruning algorithm Pool by Pool. For example, you can decide to increase Retention times for Archive or OffSite Pool.

It seems obvious to us, but apparently not to some users, that given the definition above that the Pool File and Job Retention periods is a global override for the normal Client based pruning, which means that when the Job is pruned, the pruning will apply globally to that particular Job.

Currently, there is a bug in the implementation that causes any Pool retention periods specified to apply to **all** Pools for that particular Client. Thus we suggest that you avoid using these two directives until this implementation problem is corrected.

11.5.8 Read-only File Daemon using capabilities

This feature implements support of keeping **ReadAll** capabilities after UID/GID switch, this allows FD to keep root read but drop write permission.

It introduces new `bacula-fd` option (`-k`) specifying that **ReadAll** capabilities should be kept after UID/GID switch.

```
root@localhost:~# bacula-fd -k -u nobody -g nobody
```

The code for this feature was contributed by our friends at AltLinux.

11.5.9 Bvfs API

To help developers of restore GUI interfaces, we have added new *dot commands* that permit browsing the catalog in a very simple way.

- `.bvfs_update [jobid=x,y,z]` This command is required to update the Bvfs cache in the catalog. You need to run it before any access to the Bvfs layer.
- `.bvfs_lsdirs jobid=x,y,z path=/path | pathid=101` This command will list all directories in the specified `path` or `pathid`. Using `pathid` avoids problems with character encoding of path/filenames.
- `.bvfs_lsfiles jobid=x,y,z path=/path | pathid=101` This command will list all files in the specified `path` or `pathid`. Using `pathid` avoids problems with character encoding.

You can use `limit=xxx` and `offset=yyy` to limit the amount of data that will be displayed.

```
* .bvfs_update jobid=1,2
* .bvfs_update
* .bvfs_lsdir path=/ jobid=1,2
```

This project was funded by Bacula Systems.

11.5.10 Testing your Tape Drive

To determine the best configuration of your tape drive, you can run the new speed command available in the btape program.

This command can have the following arguments:

`file_size=n` Specify the Maximum File Size for this test (between 1 and 5GB). This counter is in GB.

`nb_file=n` Specify the number of file to be written. The amount of data should be greater than your memory ($file\_size * nb\_file$).

`skip_zero` This flag permits to skip tests with constant data.

`skip_random` This flag permits to skip tests with random data.

`skip_raw` This flag permits to skip tests with raw access.

`skip_block` This flag permits to skip tests with Bacula block access.

```
*speed file_size=3 skip_raw
btape.c:1078 Test with zero data and bacula block structure.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 44.128 MB/s
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 43.531 MB/s

btape.c:1090 Test with random data, should give the minimum throughput.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 7.271 MB/s
++++++++++++++++++++++++++++++++++++++++
```

```
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 7.365 MB/s
```

When using compression, the random test will give your the minimum throughput of your drive . The test using constant string will give you the maximum speed of your hardware chain. (CPU, memory, SCSI card, cable, drive, tape).

You can change the block size in the Storage Daemon configuration file.

11.5.11 New Block Checksum Device Directive

You may now turn off the Block Checksum (CRC32) code that Bacula uses when writing blocks to a Volume. This is done by adding:

```
Block Checksum = no
```

doing so can reduce the Storage daemon CPU usage slightly. It will also permit Bacula to read a Volume that has corrupted data.

The default is **yes** – i.e. the checksum is computed on write and checked on read.

We do not recommend to turn this off particularly on older tape drives or for disk Volumes where doing so may allow corrupted data to go undetected.

11.5.12 New Bat Features

Those new features were funded by Bacula Systems.

Media List View

By clicking on "Media", you can see the list of all your volumes. You will be able to filter by Pool, Media Type, Location,... And sort the result directly in the table. The old "Media" view is now known as "Pool".

Media Information View

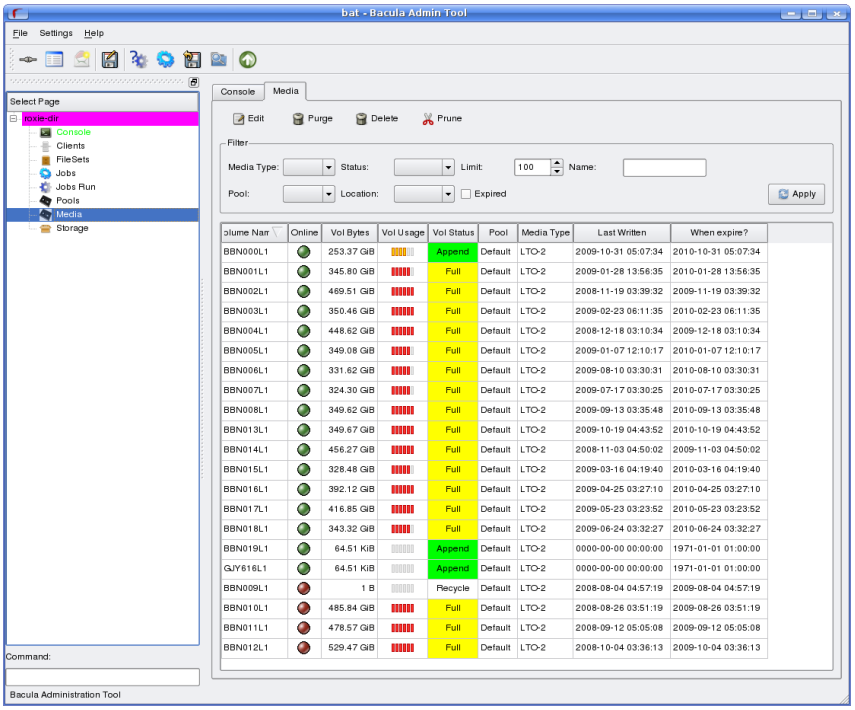
By double-clicking on a volume (on the Media list, in the Autochanger content or in the Job information panel), you can access a detailed overview of your Volume. (cf figure 11.4 on the following page.)

Job Information View

By double-clicking on a Job record (on the Job run list or in the Media information panel), you can access a detailed overview of your Job. (cf figure 11.5 on page 115.)

Autochanger Content View

By double-clicking on a Storage record (on the Storage list panel), you can access a detailed overview of your Autochanger. (cf figure 11.5 on page 115.)



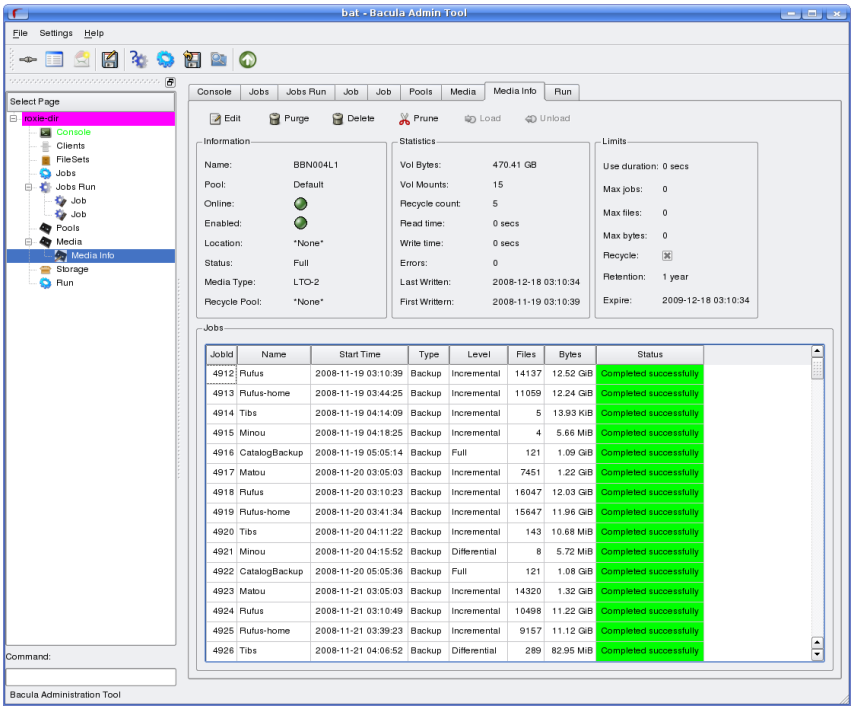


Figure 11.4: Media information

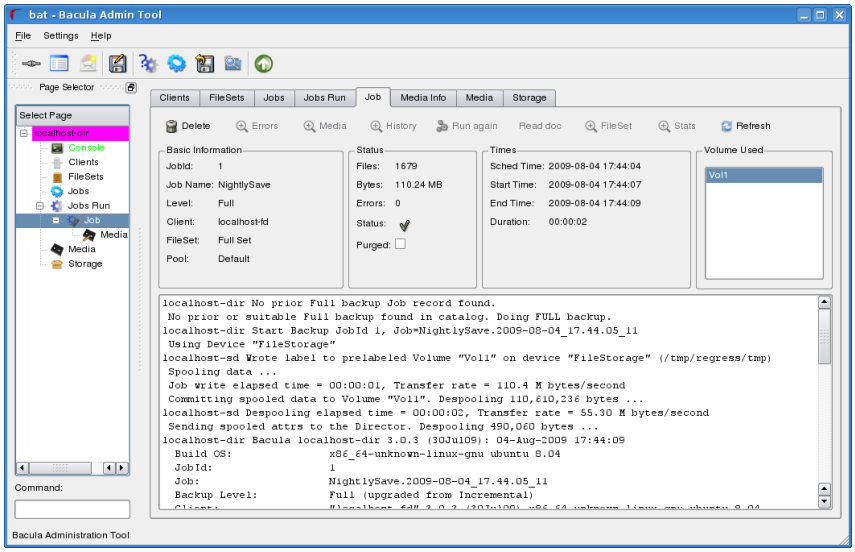


Figure 11.5: Job information

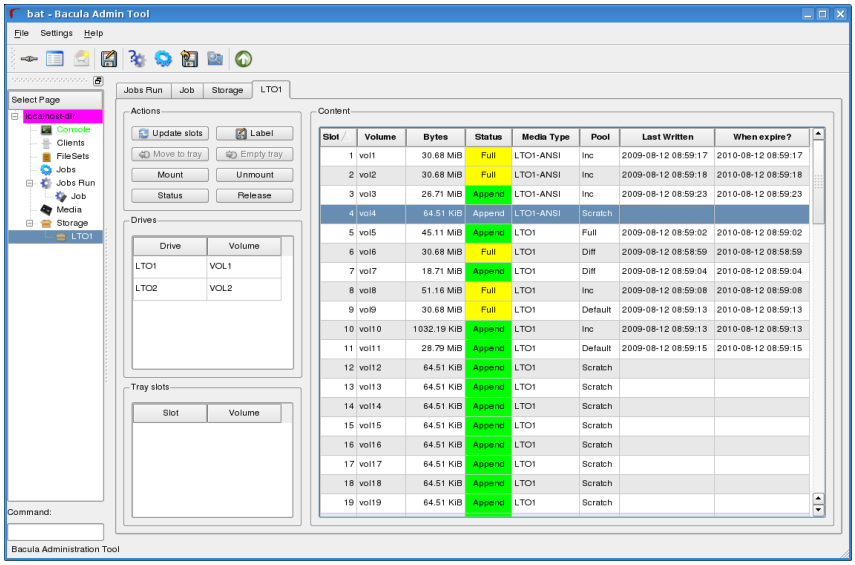


Figure 11.6: Autochanger content

To use this feature, you need to use the latest mtx-changer script version. (With new `listall` and `transfer` commands)

11.5.13 Bat on Windows

We have ported **bat** to Windows and it is now installed by default when the installer is run. It works quite well on Win32, but has not had a lot of testing there, so your feedback would be welcome. Unfortunately, even though it is installed by default, it does not yet work on 64 bit Windows operating systems.

11.5.14 New Win32 Installer

The Win32 installer has been modified in several very important ways.

- You must deinstall any current version of the Win32 File daemon before upgrading to the new one. If you forget to do so, the new installation will fail. To correct this failure, you must manually shutdown and deinstall the old File daemon.
- All files (other than menu links) are installed in **c:/Program Files/Bacula**.
- The installer no longer sets this file to require administrator privileges by default. If you want to do so, please do it manually using the **cacls** program. For example:

  ```
  cacls "C:\Program Files\Bacula" /T /G SYSTEM:F Administrators:F
  ```

- The server daemons (Director and Storage daemon) are no longer included in the Windows installer. If you want the Windows servers, you will either need to build them yourself (note they have not been ported to 64 bits), or you can contact Bacula Systems about this.

11.5.15 Win64 Installer

We have corrected a number of problems that required manual editing of the conf files. In most cases, it should now install and work. **bat** is by default installed in **c:/Program Files/Bacula/bin32** rather than **c:/Program Files/Bacula** as is the case with the 32 bit Windows installer.

11.5.16 Linux Bare Metal Recovery USB Key

We have made a number of significant improvements in the Bare Metal Recovery USB key. Please see the README files it the **rescue** release for more details.

We are working on an equivalent USB key for Windows bare metal recovery, but it will take some time to develop it (best estimate 3Q2010 or 4Q2010)

11.5.17 bconsole Timeout Option

You can now use the -u option of **bconsole** to set a timeout in seconds for commands. This is useful with GUI programs that use **bconsole** to interface to the Director.

11.5.18 Important Changes

- You are now allowed to Migrate, Copy, and Virtual Full to read and write to the same Pool. The Storage daemon ensures that you do not read and write to the same Volume.

- The `Device Poll Interval` is now 5 minutes. (previously did not poll by default).
- Virtually all the features of **mtx-changer** have now been parametrized, which allows you to configure mtx-changer without changing it. There is a new configuration file **mtx-changer.conf** that contains variables that you can set to configure mtx-changer. This configuration file will not be overwritten during upgrades. We encourage you to submit any changes that are made to mtx-changer and to parametrize it all in mtx-changer.conf so that all configuration will be done by changing only mtx-changer.conf.
- The new `mtx-changer` script has two new options, `listall` and `transfer`. Please configure them as appropriate in mtx-changer.conf.
- To enhance security of the `BackupCatalog` job, we provide a new script (`make_catalog_backup.pl`) that does not expose your catalog password. If you want to use the new script, you will need to manually change the `BackupCatalog` Job definition.
- The `bconsole help` command now accepts an argument, which if provided produces information on that command (ex: `help run`).

Truncate volume after purge

Note that the Truncate Volume after purge feature doesn't work as expected in 5.0.0 version. Please, don't use it before version 5.0.1.

Custom Catalog queries

If you wish to add specialized commands that list the contents of the catalog, you can do so by adding them to the `query.sql` file. This `query.sql` file is now empty by default. The file `examples/sample-query.sql` has an a number of sample commands you might find useful.

Deprecated parts

The following items have been **deprecated** for a long time, and are now removed from the code.

- Gnome console
- Support for SQLite. Note the source code still exists, but the project does not maintain it. Please do not use it.

11.5.19 Misc Changes

- Updated Nagios check_bacula
- Updated man files
- Added OSX package generation script in platforms/darwin
- Added Spanish and Ukrainian Bacula translations
- Enable/disable command shows only Jobs that can change
- Added `show disabled` command to show disabled Jobs
- Many ACL improvements
- Added Level to FD status Job output
- Begin Ingres DB driver (not yet working)
- Split RedHat spec files into bacula, bat, mtx, and docs
- Reorganized the manuals (fewer separate manuals)

- Added lock/unlock order protection in lock manager
- Allow 64 bit sizes for a number of variables
- Fixed several deadlocks or potential race conditions in the SD

Chapter 12

Released Version 3.0.3 and 3.0.3a

There are no new features in version 3.0.3. This version simply fixes a number of bugs found in version 3.0.2 during the ongoing development process.

12.1 New Features in Released Version 3.0.2

This chapter presents the new features added to the Released Bacula Version 3.0.2.

12.1.1 Full Restore from a Given JobId

This feature allows selecting a single JobId and having Bacula automatically select all the other jobs that comprise a full backup up to and including the selected date (through JobId).

Assume we start with the following jobs:

```
+-------+--------------+---------------------+-------+----------+------------+
| jobid | client       | starttime           | level | jobfiles | jobbytes   |
+-------+--------------+---------------------+-------+----------+------------
| 6     | localhost-fd | 2009-07-15 11:45:49 | I     | 2        | 0          |
| 5     | localhost-fd | 2009-07-15 11:45:45 | I     | 15       | 44143      |
| 3     | localhost-fd | 2009-07-15 11:45:38 | I     | 1        | 10         |
| 1     | localhost-fd | 2009-07-15 11:45:30 | F     | 1527     | 44143073   |
+-------+--------------+---------------------+-------+----------+------------+
```

Below is an example of this new feature (which is number 12 in the menu).

```
* restore
To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
...
    12: Select full restore to a specified Job date
    13: Cancel

Select item:  (1-13): 12
Enter JobId to get the state to restore: 5
Selecting jobs to build the Full state at 2009-07-15 11:45:45
You have selected the following JobIds: 1,3,5
```

```
Building directory tree for JobId(s) 1,3,5 ...  ++++++++++++++++++
1,444 files inserted into the tree.
```

This project was funded by Bacula Systems.

12.1.2 Source Address

A feature has been added which allows the administrator to specify the address from which the Director and File daemons will establish connections. This may be used to simplify system configuration overhead when working in complex networks utilizing multi-homing and policy-routing.

To accomplish this, two new configuration directives have been implemented:

```
FileDaemon {
  FDSourceAddress=10.0.1.20    # Always initiate connections from this address
}

Director {
  DirSourceAddress=10.0.1.10   # Always initiate connections from this address
}
```

Simply adding specific host routes on the OS would have an undesirable side-effect: any application trying to contact the destination host would be forced to use the more specific route possibly diverting management traffic onto a backup VLAN. Instead of adding host routes for each client connected to a multi-homed backup server (for example where there are management and backup VLANs), one can use the new directives to specify a specific source address at the application level.

Additionally, this allows the simplification and abstraction of firewall rules when dealing with a Hot-Standby director or storage daemon configuration. The Hot-standby pair may share a CARP address, which connections must be sourced from, while system services listen and act from the unique interface addresses.

This project was funded by Collaborative Fusion, Inc.

12.1.3 Show volume availability when doing restore

When doing a restore the selection dialog ends by displaying this screen:

```
  The job will require the following
   Volume(s)                 Storage(s)                SD Device(s)
   ===========================================================================
   *000741L3                  LTO-4                     LTO3
   *000866L3                  LTO-4                     LTO3
   *000765L3                  LTO-4                     LTO3
   *000764L3                  LTO-4                     LTO3
   *000756L3                  LTO-4                     LTO3
   *001759L3                  LTO-4                     LTO3
   *001763L3                  LTO-4                     LTO3
    001762L3                  LTO-4                     LTO3
    001767L3                  LTO-4                     LTO3

Volumes marked with ``*'' are online (in the autochanger).
```

This should help speed up large restores by minimizing the time spent waiting for the operator to discover that he must change tapes in the library.

This project was funded by Bacula Systems.

12.1.4 Accurate estimate command

The `estimate` command can now use the accurate code to detect changes and give a better estimation.

You can set the accurate behavior on the command line by using `accurate=yes|no` or use the Job setting as default value.

```
* estimate listing accurate=yes level=incremental job=BackupJob
```

This project was funded by Bacula Systems.

12.2 New Features in 3.0.0

This chapter presents the new features added to the development 2.5.x versions to be released as Bacula version 3.0.0 sometime in April 2009.

12.2.1 Accurate Backup

As with most other backup programs, by default Bacula decides what files to backup for Incremental and Differential backup by comparing the change (st_ctime) and modification (st_mtime) times of the file to the time the last backup completed. If one of those two times is later than the last backup time, then the file will be backed up. This does not, however, permit tracking what files have been deleted and will miss any file with an old time that may have been restored to or moved onto the client filesystem.

Accurate = <yes|no>

If the **Accurate = <yes|no>** directive is enabled (default no) in the Job resource, the job will be run as an Accurate Job. For a **Full** backup, there is no difference, but for **Differential** and **Incremental** backups, the Director will send a list of all previous files backed up, and the File daemon will use that list to determine if any new files have been added or or moved and if any files have been deleted. This allows Bacula to make an accurate backup of your system to that point in time so that if you do a restore, it will restore your system exactly.

One note of caution about using Accurate backup is that it requires more resources (CPU and memory) on both the Director and the Client machines to create the list of previous files backed up, to send that list to the File daemon, for the File daemon to keep the list (possibly very big) in memory, and for the File daemon to do comparisons between every file in the FileSet and the list. In particular, if your client has lots of files (more than a few million), you will need lots of memory on the client machine.

Accurate must not be enabled when backing up with a plugin that is not specially designed to work with Accurate. If you enable it, your restores will probably not work correctly.

This project was funded by Bacula Systems.

12.2.2 Copy Jobs

A new **Copy** job type 'C' has been implemented. It is similar to the existing Migration feature with the exception that the Job that is copied is left unchanged. This essentially creates two identical copies of the same backup. However, the copy is treated as a copy rather than a backup job, and hence is not directly available for restore. The **restore** command lists copy jobs and allows selection of copies by using `jobid=` option. If the keyword **copies** is present on the command line, Bacula will display the list of all copies for selected jobs.

```
* restore copies
[...]
These JobIds have copies as follows:
+-------+-----------------------------------+-----------+-----------------+
| JobId | Job                               | CopyJobId | MediaType       |
+-------+-----------------------------------+-----------+-----------------+
| 2     | CopyJobSave.2009-02-17_16.31.00.11 | 7         | DiskChangerMedia |
+-------+-----------------------------------+-----------+-----------------+
+-------+-------+----------+----------+---------------------+-----------------+
| JobId | Level | JobFiles | JobBytes | StartTime           | VolumeName      |
+-------+-------+----------+----------+---------------------+-----------------+
| 19    | F     | 6274     | 76565018 | 2009-02-17 16:30:45 | ChangerVolume002 |
| 2     | I     | 1        | 5        | 2009-02-17 16:30:51 | FileVolume001   |
+-------+-------+----------+----------+---------------------+-----------------+
You have selected the following JobIds: 19,2

Building directory tree for JobId(s) 19,2 ...  ++++++++++++++++++++++++++++++++++++++++
5,611 files inserted into the tree.
...
```

The Copy Job runs without using the File daemon by copying the data from the old backup Volume to a different Volume in a different Pool. See the Migration documentation for additional details. For copy Jobs there is a new selection directive named **PoolUncopiedJobs** which selects all Jobs that were not already copied to another Pool.

As with Migration, the Client, Volume, Job, or SQL query, are other possible ways of selecting the Jobs to be copied. Selection types like SmallestVolume, OldestVolume, PoolOccupancy and PoolTime also work, but are probably more suited for Migration Jobs.

If Bacula finds a Copy of a job record that is purged (deleted) from the catalog, it will promote the Copy to a *real* backup job and will make it available for automatic restore. If more than one Copy is available, it will promote the copy with the smallest JobId.

A nice solution which can be built with the new Copy feature is often called disk-to-disk-to-tape backup (DTDTT). A sample config could look something like the one below:

```
Pool {
  Name = FullBackupsVirtualPool
  Pool Type = Backup
  Purge Oldest Volume = Yes
  Storage = vtl
  NextPool = FullBackupsTapePool
}

Pool {
  Name = FullBackupsTapePool
  Pool Type = Backup
  Recycle = Yes
  AutoPrune = Yes
```

```
  Volume Retention = 365 days
  Storage = superloader
}

#
# Fake fileset for copy jobs
#
Fileset {
  Name = None
  Include {
    Options {
      signature = MD5
    }
  }
}

#
# Fake client for copy jobs
#
Client {
  Name = None
  Address = localhost
  Password = "NoNe"
  Catalog = MyCatalog
}

#
# Default template for a CopyDiskToTape Job
#
JobDefs {
  Name = CopyDiskToTape
  Type = Copy
  Messages = StandardCopy
  Client = None
  FileSet = None
  Selection Type = PoolUncopiedJobs
  Maximum Concurrent Jobs = 10
  SpoolData = No
  Allow Duplicate Jobs = Yes
  Cancel Queued Duplicates = No
  Cancel Running Duplicates = No
  Priority = 13
}

Schedule {
   Name = DaySchedule7:00
   Run = Level=Full daily at 7:00
}

Job {
  Name = CopyDiskToTapeFullBackups
  Enabled = Yes
  Schedule = DaySchedule7:00
  Pool = FullBackupsVirtualPool
  JobDefs = CopyDiskToTape
}
```

The example above had 2 pool which are copied using the PoolUncopiedJobs selection criteria. Normal Full backups go to the Virtual pool and are copied to the Tape pool the next morning.

The command `list copies [jobid=x,y,z]` lists copies for a given **jobid**.

```
*list copies
+-------+-----------------------------------+-----------+-----------------+
| JobId | Job                               | CopyJobId | MediaType       |
+-------+-----------------------------------+-----------+-----------------+
|     9 | CopyJobSave.2008-12-20_22.26.49.05 |        11 | DiskChangerMedia |
+-------+-----------------------------------+-----------+-----------------+
```

12.2.3 ACL Updates

The whole ACL code had been overhauled and in this version each platforms has different streams for each type of acl available on such an platform. As ACLs between platforms tend to be not that portable (most implement POSIX acls but some use an other draft or a completely different format) we currently only allow certain platform specific ACL streams to be decoded and restored on the same platform that they were created on. The old code allowed to restore ACL cross platform but the comments already mention that not being to wise. For backward compatibility the new code will accept the two old ACL streams and handle those with the platform specific handler. But for all new backups it will save the ACLs using the new streams.

Currently the following platforms support ACLs:

- **AIX**
- **Darwin/OSX**
- **FreeBSD**
- **HPUX**
- **IRIX**
- **Linux**
- **Tru64**
- **Solaris**

Currently we support the following ACL types (these ACL streams use a reserved part of the stream numbers):

- **STREAM_ACL_AIX_TEXT** 1000 AIX specific string representation from acl_get
- **STREAM_ACL_DARWIN_ACCESS_ACL** 1001 Darwin (OSX) specific acl_t string representation from acl_to_text (POSIX acl)
- **STREAM_ACL_FREEBSD_DEFAULT_ACL** 1002 FreeBSD specific acl_t string representation from acl_to_text (POSIX acl) for default acls.
- **STREAM_ACL_FREEBSD_ACCESS_ACL** 1003 FreeBSD specific acl_t string representation from acl_to_text (POSIX acl) for access acls.
- **STREAM_ACL_HPUX_ACL_ENTRY** 1004 HPUX specific acl_entry string representation from acltostr (POSIX acl)
- **STREAM_ACL_IRIX_DEFAULT_ACL** 1005 IRIX specific acl_t string representation from acl_to_text (POSIX acl) for default acls.
- **STREAM_ACL_IRIX_ACCESS_ACL** 1006 IRIX specific acl_t string representation from acl_to_text (POSIX acl) for access acls.
- **STREAM_ACL_LINUX_DEFAULT_ACL** 1007 Linux specific acl_t string representation from acl_to_text (POSIX acl) for default acls.

- **STREAM_ACL_LINUX_ACCESS_ACL** 1008 Linux specific acl_t string representation from acl_to_text (POSIX acl) for access acls.
- **STREAM_ACL_TRU64_DEFAULT_ACL** 1009 Tru64 specific acl_t string representation from acl_to_text (POSIX acl) for default acls.
- **STREAM_ACL_TRU64_DEFAULT_DIR_ACL** 1010 Tru64 specific acl_t string representation from acl_to_text (POSIX acl) for default acls.
- **STREAM_ACL_TRU64_ACCESS_ACL** 1011 Tru64 specific acl_t string representation from acl_to_text (POSIX acl) for access acls.
- **STREAM_ACL_SOLARIS_ACLENT** 1012 Solaris specific aclent_t string representation from acltotext or acl_totext (POSIX acl)
- **STREAM_ACL_SOLARIS_ACE** 1013 Solaris specific ace_t string representation from from acl_totext (NFSv4 or ZFS acl)

In future versions we might support conversion functions from one type of acl into an other for types that are either the same or easily convertible. For now the streams are separate and restoring them on a platform that doesn't recognize them will give you a warning.

12.2.4 Extended Attributes

Something that was on the project list for some time is now implemented for platforms that support a similar kind of interface. Its the support for backup and restore of so called extended attributes. As extended attributes are so platform specific these attributes are saved in separate streams for each platform. Restores of the extended attributes can only be performed on the same platform the backup was done. There is support for all types of extended attributes, but restoring from one type of filesystem onto an other type of filesystem on the same platform may lead to surprises. As extended attributes can contain any type of data they are stored as a series of so called value-pairs. This data must be seen as mostly binary and is stored as such. As security labels from selinux are also extended attributes this option also stores those labels and no specific code is enabled for handling selinux security labels.

Currently the following platforms support extended attributes:

- **Darwin/OSX**
- **FreeBSD**
- **Linux**
- **NetBSD**

On Linux acls are also extended attributes, as such when you enable ACLs on a Linux platform it will NOT save the same data twice e.g. it will save the ACLs and not the same extended attribute.

To enable the backup of extended attributes please add the following to your fileset definition.

```
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
      xattrsupport = yes
    }
    File = ...
  }
}
```

12.2.5 Shared objects

A default build of Bacula will now create the libraries as shared objects (.so) rather than static libraries as was previously the case. The shared libraries are built using **libtool** so it should be quite portable.

An important advantage of using shared objects is that on a machine with the Directory, File daemon, the Storage daemon, and a console, you will have only one copy of the code in memory rather than four copies. Also the total size of the binary release is smaller since the library code appears only once rather than once for every program that uses it; this results in significant reduction in the size of the binaries particularly for the utility tools.

In order for the system loader to find the shared objects when loading the Bacula binaries, the Bacula shared objects must either be in a shared object directory known to the loader (typically /usr/lib) or they must be in the directory that may be specified on the **./configure** line using the **--libdir** option as:

```
./configure --libdir=/full-path/dir
```

the default is /usr/lib. If --libdir is specified, there should be no need to modify your loader configuration provided that the shared objects are installed in that directory (Bacula does this with the make install command). The shared objects that Bacula references are:

```
libbaccfg.so
libbacfind.so
libbacpy.so
libbac.so
```

These files are symbolically linked to the real shared object file, which has a version number to permit running multiple versions of the libraries if desired (not normally the case).

If you have problems with libtool or you wish to use the old way of building static libraries, or you want to build a static version of Bacula you may disable libtool on the configure command line with:

```
./configure --disable-libtool
```

12.2.6 Building Static versions of Bacula

In order to build static versions of Bacula, in addition to configuration options that were needed you now must also add –disable-libtool. Example

```
./configure --enable-static-client-only --disable-libtool
```

12.2.7 Virtual Backup (Vbackup)

Bacula's virtual backup feature is often called Synthetic Backup or Consolidation in other backup products. It permits you to consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

In some respects the Vbackup feature works similar to a Migration job, in that Bacula normally reads the data from the pool specified in the Job resource, and writes it to the **Next Pool** specified in the Job resource. Note, this means that usually the output from the Virtual Backup is written into a different pool from where your prior backups are saved. Doing it this way guarantees that you will not get a deadlock situation attempting to read and write to the same volume in the Storage daemon. If you then want to do subsequent backups, you may need to move the Virtual Full Volume back to your normal backup pool. Alternatively, you can set your **Next Pool** to point to the current pool. This will cause Bacula to read and write to Volumes in the current pool. In general, this will work, because Bacula will not allow reading and writing on the same Volume. In any case, once a VirtualFull has been created, and a restore is done involving the most current Full, it will read the Volume or Volumes by the VirtualFull regardless of in which Pool the Volume is found.

The Vbackup is enabled on a Job by Job in the Job resource by specifying a level of **VirtualFull**.

A typical Job resource definition might look like the following:

```
Job {
  Name = "MyBackup"
  Type = Backup
  Client=localhost-fd
  FileSet = "Full Set"
  Storage = File
  Messages = Standard
  Pool = Default
  SpoolData = yes
}

# Default pool definition
Pool {
  Name = Default
  Pool Type = Backup
  Recycle = yes            # Automatically recycle Volumes
  AutoPrune = yes          # Prune expired volumes
  Volume Retention = 365d  # one year
  NextPool = Full
  Storage = File
}

Pool {
  Name = Full
  Pool Type = Backup
  Recycle = yes            # Automatically recycle Volumes
  AutoPrune = yes          # Prune expired volumes
  Volume Retention = 365d  # one year
  Storage = DiskChanger
}

# Definition of file storage device
Storage {
  Name = File
  Address = localhost
  Password = "xxx"
  Device = FileStorage
  Media Type = File
  Maximum Concurrent Jobs = 5
}

# Definition of DDS Virtual tape disk storage device
Storage {
```

```
  Name = DiskChanger
  Address = localhost  # N.B. Use a fully qualified name here
  Password = "yyy"
  Device = DiskChanger
  Media Type = DiskChangerMedia
  Maximum Concurrent Jobs = 4
  Autochanger = yes
}
```

Then in bconsole or via a Run schedule, you would run the job as:

```
run job=MyBackup level=Full
run job=MyBackup level=Incremental
run job=MyBackup level=Differential
run job=MyBackup level=Incremental
run job=MyBackup level=Incremental
```

So providing there were changes between each of those jobs, you would end up with a Full backup, a Differential, which includes the first Incremental backup, then two Incremental backups. All the above jobs would be written to the **Default** pool.

To consolidate those backups into a new Full backup, you would run the following:

```
run job=MyBackup level=VirtualFull
```

And it would produce a new Full backup without using the client, and the output would be written to the **Full** Pool which uses the Diskchanger Storage.

If the Virtual Full is run, and there are no prior Jobs, the Virtual Full will fail with an error.

Note, the Start and End time of the Virtual Full backup is set to the values for the last job included in the Virtual Full (in the above example, it is an Increment). This is so that if another incremental is done, which will be based on the Virtual Full, it will backup all files from the last Job included in the Virtual Full rather than from the time the Virtual Full was actually run.

12.2.8 Catalog Format

Bacula 3.0 comes with some changes to the catalog format. The upgrade operation will convert the FileId field of the File table from 32 bits (max 4 billion table entries) to 64 bits (very large number of items). The conversion process can take a bit of time and will likely DOUBLE THE SIZE of your catalog during the conversion. Also you won't be able to run jobs during this conversion period. For example, a 3 million file catalog will take 2 minutes to upgrade on a normal machine. Please don't forget to make a valid backup of your database before executing the upgrade script. See the ReleaseNotes for additional details.

12.2.9 64 bit Windows Client

Unfortunately, Microsoft's implementation of Volume Shadown Copy (VSS) on their 64 bit OS versions is not compatible with a 32 bit Bacula Client. As a consequence, we are also releasing a 64 bit version of the Bacula Windows Client (win64bacula-3.0.0.exe) that does work with VSS. These binaries should only be installed on 64 bit Windows operating systems. What is important is not your hardware but whether or not you have a 64 bit version of the Windows OS.

Compared to the Win32 Bacula Client, the 64 bit release contains a few differences:

1. Before installing the Win64 Bacula Client, you must totally deinstall any prior 2.4.x Client installation using the Bacula deinstallation (see the menu item). You may want to save your .conf files first.
2. Only the Client (File daemon) is ported to Win64, the Director and the Storage daemon are not in the 64 bit Windows installer.
3. bwx-console is not yet ported.
4. bconsole is ported but it has not been tested.
5. The documentation is not included in the installer.
6. Due to Vista security restrictions imposed on a default installation of Vista, before upgrading the Client, you must manually stop any prior version of Bacula from running, otherwise the install will fail.
7. Due to Vista security restrictions imposed on a default installation of Vista, attempting to edit the conf files via the menu items will fail. You must directly edit the files with appropriate permissions. Generally double clicking on the appropriate .conf file will work providing you have sufficient permissions.
8. All Bacula files are now installed in **C:/Program Files/Bacula** except the main menu items, which are installed as before. This vastly simplifies the installation.
9. If you are running on a foreign language version of Windows, most likely **C:/Program Files** does not exist, so you should use the Custom installation and enter an appropriate location to install the files.
10. The 3.0.0 Win32 Client continues to install files in the locations used by prior versions. For the next version we will convert it to use the same installation conventions as the Win64 version.

This project was funded by Bacula Systems.

12.2.10 Duplicate Job Control

The new version of Bacula provides four new directives that give additional control over what Bacula does if duplicate jobs are started. A duplicate job in the sense we use it here means a second or subsequent job with the same name starts. This happens most frequently when the first job runs longer than expected because no tapes are available.

The four directives each take as an argument a **yes** or **no** value and are specified in the Job resource.

They are:

Allow Duplicate Jobs = <**yes**|**no**>

If this directive is set to **yes**, duplicate jobs will be run. If the directive is set to **no** (default) then only one job of a given name may run at one time, and the action that Bacula takes to ensure only one job runs is determined by the other directives (see below).

If **Allow Duplicate Jobs** is set to **no** and two jobs are present and none of the three directives given below permit Canceling a job, then the current job (the second one started) will be canceled.

Allow Higher Duplicates = <**yes**|**no**>

This directive was in version 5.0.0, but does not work as expected. If used, it should always be set to no. In later versions of Bacula the directive is disabled (disregarded).

Cancel Running Duplicates = <yes|no>

If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already running will be canceled. The default is **no**.

Cancel Queued Duplicates = <yes|no>

If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already queued to run but not yet running will be canceled. The default is **no**.

12.2.11 TLS Authentication

In Bacula version 2.5.x and later, in addition to the normal Bacula CRAM-MD5 authentication that is used to authenticate each Bacula connection, you can specify that you want TLS Authentication as well, which will provide more secure authentication.

This new feature uses Bacula's existing TLS code (normally used for communications encryption) to do authentication. To use it, you must specify all the TLS directives normally used to enable communications encryption (TLS Enable, TLS Verify Peer, TLS Certificate, ...) and a new directive:

TLS Authenticate = yes

```
TLS Authenticate = yes
```

in the main daemon configuration resource (Director for the Director, Client for the File daemon, and Storage for the Storage daemon).

When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula daemons will be done without encryption.

If you want to encrypt communications data, use the normal TLS directives but do not turn on **TLS Authenticate**.

12.2.12 bextract non-portable Win32 data

bextract has been enhanced to be able to restore non-portable Win32 data to any OS. Previous versions were unable to restore non-portable Win32 data to machines that did not have the Win32 BackupRead and BackupWrite API calls.

12.2.13 State File updated at Job Termination

In previous versions of Bacula, the state file, which provides a summary of previous jobs run in the **status** command output was updated only when Bacula terminated, thus if the daemon crashed, the state file might not contain all the run data. This version of the Bacula daemons updates the state file on each job termination.

12.2.14 MaxFullInterval = <time-interval>

The new Job resource directive **Max Full Interval = <time-interval>** can be used to specify the maximum time interval between **Full** backup jobs. When a job starts, if the time since the last Full backup is greater than the specified interval, and the job would normally be an **Incremental** or **Differential**, it will be automatically upgraded to a **Full** backup.

12.2.15 MaxDiffInterval = <time-interval>

The new Job resource directive **Max Diff Interval = <time-interval>** can be used to specify the maximum time interval between **Differential** backup jobs. When a job starts, if the time since the last Differential backup is greater than the specified interval, and the job would normally be an **Incremental**, it will be automatically upgraded to a **Differential** backup.

12.2.16 Honor No Dump Flag = <yes|no>

On FreeBSD systems, each file has a **no dump flag** that can be set by the user, and when it is set it is an indication to backup programs to not backup that particular file. This version of Bacula contains a new Options directive within a FileSet resource, which instructs Bacula to obey this flag. The new directive is:

```
Honor No Dump Flag = yes\vb{}no
```

The default value is **no**.

12.2.17 Exclude Dir Containing = <filename-string>

The **ExcludeDirContaining = <filename>** is a new directive that can be added to the Include section of the FileSet resource. If the specified filename (**filename-string**) is found on the Client in any directory to be backed up, the whole directory will be ignored (not backed up). For example:

```
# List of files to be backed up
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
    }
    File = /home
    Exclude Dir Containing = .excludeme
  }
}
```

But in /home, there may be hundreds of directories of users and some people want to indicate that they don't want to have certain directories backed up. For example, with the above FileSet, if the user or sysadmin creates a file named **.excludeme** in specific directories, such as

```
/home/user/www/cache/.excludeme
/home/user/temp/.excludeme
```

then Bacula will not backup the two directories named:

```
/home/user/www/cache
/home/user/temp
```

NOTE: subdirectories will not be backed up. That is, the directive applies to the two directories in question and any children (be they files, directories, etc).

12.2.18 Bacula Plugins

Support for shared object plugins has been implemented in the Linux, Unix and Win32 File daemons. The API will be documented separately in the Developer's Guide or in a new document. For the moment, there is a single plugin named **bpipe** that allows an external program to get control to backup and restore a file.

Plugins are also planned (partially implemented) in the Director and the Storage daemon.

Plugin Directory

Each daemon (DIR, FD, SD) has a new **Plugin Directory** directive that may be added to the daemon definition resource. The directory takes a quoted string argument, which is the name of the directory in which the daemon can find the Bacula plugins. If this directive is not specified, Bacula will not load any plugins. Since each plugin has a distinctive name, all the daemons can share the same plugin directory.

Plugin Options

The **Plugin Options** directive takes a quoted string argument (after the equal sign) and may be specified in the Job resource. The options specified will be passed to all plugins when they are run. This each plugin must know what it is looking for. The value defined in the Job resource can be modified by the user when he runs a Job via the **bconsole** command line prompts.

Note: this directive may be specified, and there is code to modify the string in the run command, but the plugin options are not yet passed to the plugin (i.e. not fully implemented).

Plugin Options ACL

The **Plugin Options ACL** directive may be specified in the Director's Console resource. It functions as all the other ACL commands do by permitting users running restricted consoles to specify a **Plugin Options** that overrides the one specified in the Job definition. Without this directive restricted consoles may not modify the Plugin Options.

Plugin = <plugin-command-string>

The **Plugin** directive is specified in the Include section of a FileSet resource where you put your **File = xxx** directives. For example:

```
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
    }
```

```
      File = /home
      Plugin = "bpipe:..."
    }
  }
```

In the above example, when the File daemon is processing the directives in the Include section, it will first backup all the files in **/home** then it will load the plugin named **bpipe** (actually bpipe-dir.so) from the Plugin Directory. The syntax and semantics of the Plugin directive require the first part of the string up to the colon (:) to be the name of the plugin. Everything after the first colon is ignored by the File daemon but is passed to the plugin. Thus the plugin writer may define the meaning of the rest of the string as he wishes.

Please see the next section for information about the **bpipe** Bacula plugin.

12.2.19 The bpipe Plugin

The **bpipe** plugin is provided in the directory src/plugins/fd/bpipe-fd.c of the Bacula source distribution. When the plugin is compiled and linking into the resulting dynamic shared object (DSO), it will have the name **bpipe-fd.so**. Please note that this is a very simple plugin that was written for demonstration and test purposes. It is and can be used in production, but that was never really intended.

The purpose of the plugin is to provide an interface to any system program for backup and restore. As specified above the **bpipe** plugin is specified in the Include section of your Job's FileSet resource. The full syntax of the plugin directive as interpreted by the **bpipe** plugin (each plugin is free to specify the sytax as it wishes) is:

```
Plugin = "<field1>:<field2>:<field3>:<field4>"
```

where

field1 is the name of the plugin with the trailing **-fd.so** stripped off, so in this case, we would put **bpipe** in this field.

field2 specifies the namespace, which for **bpipe** is the pseudo path and filename under which the backup will be saved. This pseudo path and filename will be seen by the user in the restore file tree. For example, if the value is **/MYSQL/regress.sql**, the data backed up by the plugin will be put under that "pseudo" path and filename. You must be careful to choose a naming convention that is unique to avoid a conflict with a path and filename that actually exists on your system.

field3 for the **bpipe** plugin specifies the "reader" program that is called by the plugin during backup to read the data. **bpipe** will call this program by doing a **popen** on it.

field4 for the **bpipe** plugin specifies the "writer" program that is called by the plugin during restore to write the data back to the filesystem.

Please note that for two items above describing the "reader" and "writer" fields, these programs are "executed" by Bacula, which means there is no shell interpretation of any command line arguments you might use. If you want to use shell characters (redirection of input or output, ...), then we recommend that you put your command or commands in a shell script and execute the script. In addition if you backup a file with the reader program, when running the writer program during the restore, Bacula will not automatically create the path to the file. Either the path must exist, or you must explicitly do so with your command or in a shell script.

Putting it all together, the full plugin directive line might look like the following:

```
Plugin = "bpipe:/MYSQL/regress.sql:mysqldump -f
          --opt --databases bacula:mysql"
```

The directive has been split into two lines, but within the **bacula-dir.conf** file would be written on a single line.

This causes the File daemon to call the **bpipe** plugin, which will write its data into the "pseudo" file **/MYSQL/regress.sql** by calling the program **mysqldump -f –opt –database bacula** to read the data during backup. The mysqldump command outputs all the data for the database named **bacula**, which will be read by the plugin and stored in the backup. During restore, the data that was backed up will be sent to the program specified in the last field, which in this case is **mysql**. When **mysql** is called, it will read the data sent to it by the plugn then write it back to the same database from which it came (**bacula** in this case).

The **bpipe** plugin is a generic pipe program, that simply transmits the data from a specified program to Bacula for backup, and then from Bacula to a specified program for restore.

By using different command lines to **bpipe**, you can backup any kind of data (ASCII or binary) depending on the program called.

12.2.20 Microsoft Exchange Server 2003/2007 Plugin

Background

The Exchange plugin was made possible by a funded development project between Equiinet Ltd – www.equiinet.com (many thanks) and Bacula Systems. The code for the plugin was written by James Harper, and the Bacula core code by Kern Sibbald. All the code for this funded development has become part of the Bacula project. Thanks to everyone who made it happen.

Concepts

Although it is possible to backup Exchange using Bacula VSS the Exchange plugin adds a good deal of functionality, because while Bacula VSS completes a full backup (snapshot) of Exchange, it does not support Incremental or Differential backups, restoring is more complicated, and a single database restore is not possible.

Microsoft Exchange organises its storage into Storage Groups with Databases inside them. A default installation of Exchange will have a single Storage Group called 'First Storage Group', with two Databases inside it, "Mailbox Store (SERVER NAME)" and "Public Folder Store (SERVER NAME)", which hold user email and public folders respectively.

In the default configuration, Exchange logs everything that happens to log files, such that if you have a backup, and all the log files since, you can restore to the present time. Each Storage Group has its own set of log files and operates independently of any other Storage Groups. At the Storage Group level, the logging can be turned off by enabling a function called "Enable circular logging". At this time the Exchange plugin will not function if this option is enabled.

The plugin allows backing up of entire storage groups, and the restoring of entire storage groups or individual databases. Backing up and restoring at the individual mailbox or email item is not supported but can be simulated by use of the "Recovery" Storage Group (see below).

Installing

The Exchange plugin requires a DLL that is shipped with Microsoft Exchanger Server called **esebcli2.dll**. Assuming Exchange is installed correctly the Exchange plugin should find this automatically and run without any additional installation.

If the DLL can not be found automatically it will need to be copied into the Bacula installation directory (eg C:\Program Files\Bacula\bin). The Exchange API DLL is named esebcli2.dll and is found in C:\Program Files\Exchsrvr\bin on a default Exchange installation.

Backing Up

To back up an Exchange server the Fileset definition must contain at least **Plugin = "exchange:/@EXCHANGE/Microsoft Information Store"** for the backup to work correctly. The 'exchange:' bit tells Bacula to look for the exchange plugin, the '@EXCHANGE' bit makes sure all the backed up files are prefixed with something that isn't going to share a name with something outside the plugin, and the 'Microsoft Information Store' bit is required also. It is also possible to add the name of a storage group to the "Plugin =" line, eg
Plugin = "exchange:/@EXCHANGE/Microsoft Information Store/First Storage Group" if you want only a single storage group backed up.

Additionally, you can suffix the 'Plugin =' directive with ":notrunconfull" which will tell the plugin not to truncate the Exchange database at the end of a full backup.

An Incremental or Differential backup will backup only the database logs for each Storage Group by inspecting the "modified date" on each physical log file. Because of the way the Exchange API works, the last logfile backed up on each backup will always be backed up by the next Incremental or Differential backup too. This adds 5MB to each Incremental or Differential backup size but otherwise does not cause any problems.

By default, a normal VSS fileset containing all the drive letters will also back up the Exchange databases using VSS. This will interfere with the plugin and Exchange's shared ideas of when the last full backup was done, and may also truncate log files incorrectly. It is important, therefore, that the Exchange database files be excluded from the backup, although the folders the files are in should be included, or they will have to be recreated manually if a bare metal restore is done.

```
FileSet {
   Include {
      File = C:/Program Files/Exchsrvr/mdbdata
      Plugin = "exchange:..."
   }
   Exclude {
      File = C:/Program Files/Exchsrvr/mdbdata/E00.chk
      File = C:/Program Files/Exchsrvr/mdbdata/E00.log
      File = C:/Program Files/Exchsrvr/mdbdata/E000000F.log
      File = C:/Program Files/Exchsrvr/mdbdata/E0000010.log
      File = C:/Program Files/Exchsrvr/mdbdata/E0000011.log
      File = C:/Program Files/Exchsrvr/mdbdata/E00tmp.log
      File = C:/Program Files/Exchsrvr/mdbdata/priv1.edb
   }
}
```

The advantage of excluding the above files is that you can significantly reduce the size of your backup since all the important Exchange files will be properly saved by the Plugin.

Restoring

The restore operation is much the same as a normal Bacula restore, with the following provisos:

- The **Where** restore option must not be specified
- Each Database directory must be marked as a whole. You cannot just select (say) the .edb file and not the others.

- If a Storage Group is restored, the directory of the Storage Group must be marked too.
- It is possible to restore only a subset of the available log files, but they **must** be contiguous. Exchange will fail to restore correctly if a log file is missing from the sequence of log files
- Each database to be restored must be dismounted and marked as "Can be overwritten by restore"
- If an entire Storage Group is to be restored (eg all databases and logs in the Storage Group), then it is best to manually delete the database files from the server (eg C:\Program Files\Exchsrvr\mdbdata*) as Exchange can get confused by stray log files lying around.

Restoring to the Recovery Storage Group

The concept of the Recovery Storage Group is well documented by Microsoft, but to briefly summarize...

Microsoft Exchange allows the creation of an additional Storage Group called the Recovery Storage Group, which is used to restore an older copy of a database (e.g. before a mailbox was deleted) into without messing with the current live data. This is required as the Standard and Small Business Server versions of Exchange can not ordinarily have more than one Storage Group.

To create the Recovery Storage Group, drill down to the Server in Exchange System Manager, right click, and select **"New -> Recovery Storage Group..."**. Accept or change the file locations and click OK. On the Recovery Storage Group, right click and select **"Add Database to Recover..."** and select the database you will be restoring.

Restore only the single database nominated as the database in the Recovery Storage Group. Exchange will redirect the restore to the Recovery Storage Group automatically. Then run the restore.

Restoring on Microsoft Server 2007

Apparently the **Exmerge** program no longer exists in Microsoft Server 2007, and hence you use a new procedure for recovering a single mail box.

This procedure is documented by Microsoft, and involves using the **Restore-Mailbox** and **Get-Mailbox Statistics** shell commands.

Caveats

This plugin is still being developed, so you should consider it currently in BETA test, and thus use in a production environment should be done only after very careful testing.

When doing a full backup, the Exchange database logs are truncated by Exchange as soon as the plugin has completed the backup. If the data never makes it to the backup medium (eg because of spooling) then the logs will still be truncated, but they will also not have been backed up. A solution to this is being worked on. You will have to schedule a new Full backup to ensure that your next backups will be usable.

The "Enable Circular Logging" option cannot be enabled or the plugin will fail.

Exchange insists that a successful Full backup must have taken place if an Incremental or Differential backup is desired, and the plugin will fail if this is not the case. If a restore is done, Exchange will require that a Full backup be done before an Incremental or Differential backup is done.

The plugin will most likely not work well if another backup application (eg NTBACKUP) is backing up the Exchange database, especially if the other backup application is truncating the log files.

The Exchange plugin has not been tested with the **Accurate** option, so we recommend either carefully testing or that you avoid this option for the current time.

The Exchange plugin is not called during processing the bconsole **estimate** command, and so anything that would be backed up by the plugin will not be added to the estimate total that is displayed.

12.2.21 libdbi Framework

As a general guideline, Bacula has support for a few catalog database drivers (MySQL and PostgreSQL) coded natively by the Bacula team. With the libdbi implementation, which is a Bacula driver that uses libdbi to access the catalog, we have an open field to use many different kinds database engines following the needs of users.

The according to libdbi (http://libdbi.sourceforge.net/) project: libdbi implements a database-independent abstraction layer in C, similar to the DBI/DBD layer in Perl. Writing one generic set of code, programmers can leverage the power of multiple databases and multiple simultaneous database connections by using this framework.

Currently the libdbi driver in Bacula project only supports the same drivers natively coded in Bacula. However the libdbi project has support for many others database engines. You can view the list at http://libdbi-drivers.sourceforge.net/. In the future all those drivers can be supported by Bacula, however, they must be tested properly by the Bacula team.

Some of benefits of using libdbi are:

- The possibility to use proprietary databases engines in which your proprietary licenses prevent the Bacula team from developing the driver.
- The possibility to use the drivers written for the libdbi project.
- The possibility to use other database engines without recompiling Bacula to use them. Just change one line in bacula-dir.conf
- Abstract Database access, this is, unique point to code and profiling catalog database access.

The following drivers have been tested:

- PostgreSQL, with and without batch insert
- Mysql, with and without batch insert

In the future, we will test and approve to use others databases engines (proprietary or not) like DB2, Oracle, Microsoft SQL.

To compile Bacula to support libdbi we need to configure the code with the –with-dbi and –with-dbi-driver=[database] ./configure options, where [database] is the database engine to be used with Bacula (of course we can change the driver in file bacula-dir.conf, see below). We must configure the access port of the database engine with the option –with-db-port, because the libdbi framework doesn't know the default access port of each database.

The next phase is checking (or configuring) the bacula-dir.conf, example:

```
Catalog {
  Name = MyCatalog
```

```
  dbdriver = dbi:mysql; dbaddress = 127.0.0.1; dbport = 3306
  dbname = regress; user = regress; password = ""
}
```

The parameter **dbdriver** indicates that we will use the driver dbi with a mysql database. Currently the drivers supported by Bacula are: postgresql, amd mysql; these are the names that may be added to string "dbi:".

The following limitations apply when Bacula is set to use the libdbi framework: - Not tested on the Win32 platform - A little performance is lost if comparing with native database driver. The reason is bound with the database driver provided by libdbi and the simple fact that one more layer of code was added.

It is important to remember, when compiling Bacula with libdbi, the following packages are needed:

- libdbi version 1.0.0, http://libdbi.sourceforge.net/
- libdbi-drivers 1.0.0, http://libdbi-drivers.sourceforge.net/

You can download them and compile them on your system or install the packages from your OS distribution.

12.2.22 Console Command Additions and Enhancements

Display Autochanger Content

The **status slots storage=<storage-name>** command displays autochanger content.

```
 Slot |  Volume Name  |  Status  |  Media Type       |   Pool     |
------+---------------+----------+-------------------+------------|
    1 |         00001 |   Append |  DiskChangerMedia |    Default |
    2 |         00002 |   Append |  DiskChangerMedia |    Default |
    3*|         00003 |   Append |  DiskChangerMedia |    Scratch |
    4 |               |          |                   |            |
```

If you an asterisk (*****) appears after the slot number, you must run an **update slots** command to synchronize autochanger content with your catalog.

list joblog job=xxx or jobid=nnn

A new list command has been added that allows you to list the contents of the Job Log stored in the catalog for either a Job Name (fully qualified) or for a particular JobId. The **llist** command will include a line with the time and date of the entry.

Note for the catalog to have Job Log entries, you must have a directive such as:

```
  catalog = all
```

In your Director's **Messages** resource.

Use separator for multiple commands

When using bconsole with readline, you can set the command separator with **@separator** command to one of those characters to write commands who require multiple input in one line.

```
!$%&'()*+,-/:;<>?[]^`{|}~
```

Deleting Volumes

The delete volume bconsole command has been modified to require an asterisk (*) in front of a MediaId otherwise the value you enter is a taken to be a Volume name. This is so that users may delete numeric Volume names. The previous Bacula versions assumed that all input that started with a number was a MediaId.

This new behavior is indicated in the prompt if you read it carefully.

12.2.23 Bare Metal Recovery

The old bare metal recovery project is essentially dead. One of the main features of it was that it would build a recovery CD based on the kernel on your system. The problem was that every distribution has a different boot procedure and different scripts, and worse yet, the boot procedures and scripts change from one distribution to another. This meant that maintaining (keeping up with the changes) the rescue CD was too much work.

To replace it, a new bare metal recovery USB boot stick has been developed by Bacula Systems. This technology involves remastering a Ubuntu LiveCD to boot from a USB key.

Advantages:

1. Recovery can be done from within graphical environment.
2. Recovery can be done in a shell.
3. Ubuntu boots on a large number of Linux systems.
4. The process of updating the system and adding new packages is not too difficult.
5. The USB key can easily be upgraded to newer Ubuntu versions.
6. The USB key has writable partitions for modifications to the OS and for modification to your home directory.
7. You can add new files/directories to the USB key very easily.
8. You can save the environment from multiple machines on one USB key.
9. Bacula Systems is funding its ongoing development.

The disadvantages are:

1. The USB key is usable but currently under development.
2. Not everyone may be familiar with Ubuntu (no worse than using Knoppix)
3. Some older OSes cannot be booted from USB. This can be resolved by first booting a Ubuntu LiveCD then plugging in the USB key.
4. Currently the documentation is sketchy and not yet added to the main manual. See below ...

The documentation and the code can be found in the **rescue** package in the directory **linux/usb**.

12.2.24 Miscellaneous

Allow Mixed Priority = <yes|no>

This directive is only implemented in version 2.5 and later. When set to **yes** (default **no**), this job may run even if lower priority jobs are already running. This means a high priority job will not have to wait for other jobs to finish before starting. The scheduler will only mix priorities when all running jobs have this set to true.

Note that only higher priority jobs will start early. Suppose the director will allow two concurrent jobs, and that two jobs with priority 10 are running, with two more in the queue. If a job with priority 5 is added to the queue, it will be run as soon as one of the running jobs finishes. However, new priority 10 jobs will not be run until the priority 5 job has finished.

Bootstrap File Directive – FileRegex

FileRegex is a new command that can be added to the bootstrap (.bsr) file. The value is a regular expression. When specified, only matching filenames will be restored.

During a restore, if all File records are pruned from the catalog for a Job, normally Bacula can restore only all files saved. That is there is no way using the catalog to select individual files. With this new feature, Bacula will ask if you want to specify a Regex expression for extracting only a part of the full backup.

```
Building directory tree for JobId(s) 1,3 ...
There were no files inserted into the tree, so file selection
is not possible.Most likely your retention policy pruned the files

Do you want to restore all the files? (yes\vb{}no): no

Regexp matching files to restore? (empty to abort): /tmp/regress/(bin|tests)/
Bootstrap records written to /tmp/regress/working/zog4-dir.restore.1.bsr
```

Bootstrap File Optimization Changes

In order to permit proper seeking on disk files, we have extended the bootstrap file format to include a **VolStartAddr** and **VolEndAddr** records. Each takes a 64 bit unsigned integer range (i.e. nnn-mmm) which defines the start address range and end address range respectively. These two directives replace the **VolStartFile**, **VolEndFile**, **VolStartBlock** and **VolEndBlock** directives. Bootstrap files containing the old directives will still work, but will not properly take advantage of proper disk seeking, and may read completely to the end of a disk volume during a restore. With the new format (automatically generated by the new Director), restores will seek properly and stop reading the volume when all the files have been restored.

Solaris ZFS/NFSv4 ACLs

This is an upgrade of the previous Solaris ACL backup code to the new library format, which will backup both the old POSIX(UFS) ACLs as well as the ZFS ACLs.

The new code can also restore POSIX(UFS) ACLs to a ZFS filesystem (it will translate the POSIX(UFS)) ACL into a ZFS/NFSv4 one) it can also be used to transfer from UFS to ZFS filesystems.

Virtual Tape Emulation

We now have a Virtual Tape emulator that allows us to run though 99.9% of the tape code but actually reading and writing to a disk file. Used with the **disk-changer** script, you can now emulate an autochanger with 10 drives and 700 slots. This feature is most useful in testing. It is enabled by using **Device Type = vtape** in the Storage daemon's Device directive. This feature is only implemented on Linux machines and should not be used for production.

Bat Enhancements

Bat (the Bacula Administration Tool) GUI program has been significantly enhanced and stabilized. In particular, there are new table based status commands; it can now be easily localized using Qt4 Linguist.

The Bat communications protocol has been significantly enhanced to improve GUI handling. Note, you **must** use a the bat that is distributed with the Director you are using otherwise the communications protocol will not work.

RunScript Enhancements

The **RunScript** resource has been enhanced to permit multiple commands per RunScript. Simply specify multiple **Command** directives in your RunScript.

```
Job {
  Name = aJob
  RunScript {
    Command = "/bin/echo test"
    Command = "/bin/echo an other test"
    Command = "/bin/echo 3 commands in the same runscript"
    RunsWhen = Before
  }
 ...
}
```

A new Client RunScript **RunsWhen** keyword of **AfterVSS** has been implemented, which runs the command after the Volume Shadow Copy has been made.

Console commands can be specified within a RunScript by using: **Console =** <**command**>, however, this command has not been carefully tested and debugged and is known to easily crash the Director. We would appreciate feedback. Due to the recursive nature of this command, we may remove it before the final release.

Status Enhancements

The bconsole **status dir** output has been enhanced to indicate Storage daemon job spooling and despooling activity.

Connect Timeout

The default connect timeout to the File daemon has been set to 3 minutes. Previously it was 30 minutes.

ftruncate for NFS Volumes

If you write to a Volume mounted by NFS (say on a local file server), in previous Bacula versions, when the Volume was recycled, it was not properly truncated because NFS does not implement ftruncate (file truncate). This is now corrected in the new version because we have written code (actually a kind user) that deletes and recreates the Volume, thus accomplishing the same thing as a truncate.

Support for Ubuntu

The new version of Bacula now recognizes the Ubuntu (and Kubuntu) version of Linux, and thus now provides correct autostart routines. Since Ubuntu officially supports Bacula, you can also obtain any recent release of Bacula from the Ubuntu repositories.

Recycle Pool = <pool-name>

The new **RecyclePool** directive defines to which pool the Volume will be placed (moved) when it is recycled. Without this directive, a Volume will remain in the same pool when it is recycled. With this directive, it can be moved automatically to any existing pool during a recycle. This directive is probably most useful when defined in the Scratch pool, so that volumes will be recycled back into the Scratch pool.

FD Version

The File daemon to Director protocol now includes a version number, which although there is no visible change for users, will help us in future versions automatically determine if a File daemon is not compatible.

Max Run Sched Time = <time-period-in-seconds>

The time specifies the maximum allowed time that a job may run, counted from when the job was scheduled. This can be useful to prevent jobs from running during working hours. We can see it like `Max Start Delay + Max Run Time`.

Max Wait Time = <time-period-in-seconds>

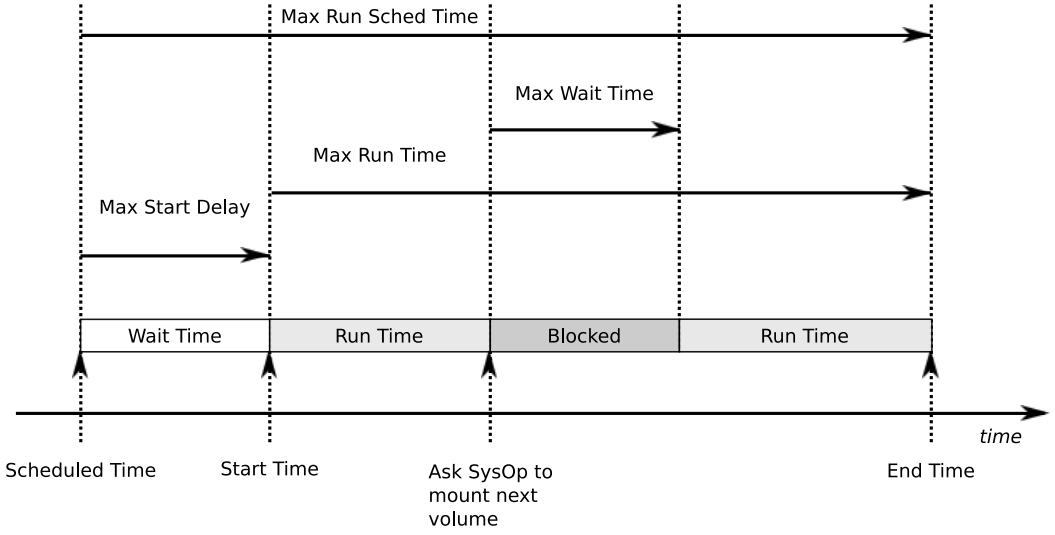
Previous **MaxWaitTime** directives aren't working as expected, instead of checking the maximum allowed time that a job may block for a resource, those directives worked like **MaxRunTime**. Some users are reporting to use **Incr/Diff/Full Max Wait Time** to control the maximum run time of their job depending on the level. Now, they have to use **Incr/Diff/Full Max Run Time**. **Incr/Diff/Full Max Wait Time** directives are now deprecated.

Incremental|Differential Max Wait Time = <time-period-in-seconds>

These directives have been deprecated in favor of `Incremental|Differential Max Run Time`.

Max Run Time directives

Using **Full/Diff/Incr Max Run Time**, it's now possible to specify the maximum allowed time that a job can run depending on the level.



Statistics Enhancements

If you (or probably your boss) want to have statistics on your backups to provide some *Service Level Agreement* indicators, you could use a few SQL queries on the Job table to report how many:

- jobs have run
- jobs have been successful
- files have been backed up
- ...

However, these statistics are accurate only if your job retention is greater than your statistics period. Ie, if jobs are purged from the catalog, you won't be able to use them.

Now, you can use the **update stats [days=num]** console command to fill the JobHistory table with new Job records. If you want to be sure to take in account only **good jobs**, ie if one of your important job has failed but you have fixed the problem and restarted it on time, you probably want to delete the first *bad* job record and keep only the successful one. For that simply let your staff do the job, and update JobHistory table after two or three days depending on your organization using the **[days=num]** option.

These statistics records aren't used for restoring, but mainly for capacity planning, billings, etc.

The Bweb interface provides a statistics module that can use this feature. You can also use tools like Talend or extract information by yourself.

The **Statistics Retention = <time>** director directive defines the length of time that Bacula will keep statistics job records in the Catalog database after the Job End time. (In `JobHistory` table) When this time period expires, and if user runs `prune stats` command, Bacula will prune (remove) Job records that are older than the specified period.

You can use the following Job resource in your nightly **BackupCatalog** job to maintain statistics.

```
Job {
  Name = BackupCatalog
  ...
  RunScript {
    Console = "update stats days=3"
    Console = "prune stats yes"
    RunsWhen = After
    RunsOnClient = no
  }
}
```

ScratchPool = <pool-resource-name>

This directive permits to specify a specific *Scratch* pool for the current pool. This is useful when using multiple storage sharing the same mediatype or when you want to dedicate volumes to a particular set of pool.

Enhanced Attribute Despooling

If the storage daemon and the Director are on the same machine, the spool file that contains attributes is read directly by the Director instead of being transmitted across the network. That should reduce load and speedup insertion.

SpoolSize = <size-specification-in-bytes>

A new Job directive permits to specify the spool size per job. This is used in advanced job tunning. **SpoolSize=***bytes*

MaximumConsoleConnections = <number>

A new director directive permits to specify the maximum number of Console Connections that could run concurrently. The default is set to 20, but you may set it to a larger number.

VerId = <string>

A new director directive permits to specify a personnal identifier that will be displayed in the `version` command.

dbcheck enhancements

If you are using Mysql, dbcheck will now ask you if you want to create temporary indexes to speed up orphaned Path and Filename elimination.

A new `-B` option allows you to print catalog information in a simple text based format. This is useful to backup it in a secure way.

```
$ dbcheck -B
catalog=MyCatalog
db_type=MySQL
db_name=regress
db_driver=
```

```
db_user=regress
db_password=
db_address=
db_port=0
db_socket=
```

You can now specify the database connection port in the command line.

--docdir configure option

You can use --docdir= on the ./configure command to specify the directory where you want Bacula to install the LICENSE, ReleaseNotes, ChangeLog, ... files. The default is **/usr/share/doc/bacula**.

--htmldir configure option

You can use --htmldir= on the ./configure command to specify the directory where you want Bacula to install the bat html help files. The default is **/usr/share/doc/bacula/html**

--with-plugindir configure option

You can use --plugindir= on the ./configure command to specify the directory where you want Bacula to install the plugins (currently only bpipe-fd). The default is /usr/lib.

Chapter 13

The Current State of Bacula

In other words, what is and what is not currently implemented and functional.

13.1 What is Implemented

- Job Control
  - Network backup/restore with centralized Director.
  - Internal scheduler for automatic Job execution.
  - Scheduling of multiple Jobs at the same time.
  - You may run one Job at a time or multiple simultaneous Jobs (sometimes called multiplexing).
  - Job sequencing using priorities.
  - Console interface to the Director allowing complete control. A shell, Qt5 GUI, and Web versions of the Console program are available. Note, the Qt5 GUI program called the BAT, offers many additional features over the shell program.
- Security
  - Verification of files previously cataloged, permitting a Tripwire like capability (system break-in detection).
  - CRAM-MD5 password authentication between each component (daemon).
  - Configurable TLS (SSL) communications encryption between each component.
  - Configurable Data (on Volume) encryption on a Client by Client basis.
  - Computation of MD5, SHA1 signatures of the file data if requested.
- Restore Features
  - Restore of one or more files selected interactively either for the current backup or a backup prior to a specified time and date.
  - Restore of a complete system starting from bare metal. This is mostly automated for Linux systems and partially automated for Solaris. See Disaster Recovery Using Bacula. This is also reported to work on Win2K/XP systems.
  - Listing and Restoration of files using stand-alone `bls` and `bextract` tool programs. Among other things, this permits extraction of files when Bacula and/or the catalog are not available. Note, the recommended way to restore files is using the `restore` command in the Console. These programs are designed for use as a last resort.
  - Ability to restore the catalog database rapidly by using bootstrap files (previously saved).

  - Ability to recreate the catalog database by scanning backup Volumes using the bscan program.

- SQL Catalog
  - Catalog database facility for remembering Volumes, Pools, Jobs, and Files backed up.
  - Support for MySQL and PostgreSQL Catalog databases. SQLite is deprecated and will ultimately be removed. We strongly recommend that you not use SQLite .
  - User extensible queries to the MySQL and PostgreSQL databases.
- Advanced Volume and Pool Management
  - Labeled Volumes, preventing accidental overwriting (at least by Bacula).
  - Any number of Jobs and Clients can be backed up to a single Volume. That is, you can backup and restore Linux, Unix, Sun, and Windows machines to the same Volume.
  - Multi-volume saves. When a Volume is full, **Bacula** automatically requests the next Volume and continues the backup.
  - Pool and Volume library management providing Volume flexibility (e.g. monthly, weekly, daily Volume sets, Volume sets segregated by Client, ...).
  - Machine independent Volume data format. Linux, Solaris, and Windows clients can all be backed up to the same Volume if desired.
  - The Volume data format is upwards compatible so that old Volumes can always be read.
  - A flexible message handler including routing of messages from any daemon back to the Director and automatic email reporting.
  - Data spooling to disk during backup with subsequent write to tape from the spooled disk files. This prevents tape "shoe shine" during Incremental/Differential backups.
- Advanced Support for most Storage Devices
  - Autochanger support using a simple shell interface that can interface to virtually any autoloader program. A script for mtx is provided.
  - Support for autochanger barcodes – automatic tape labeling from barcodes.
  - Automatic support for multiple autochanger magazines either using barcodes or by reading the tapes.
  - Support for multiple drive autochangers.
  - Raw device backup/restore. Restore must be to the same device.
  - All Volume blocks (approximately 64K bytes) contain a data checksum.
  - Migration support – move data from one Pool to another or one Volume to another.
  - Supports writing to DVD.
- Multi-Operating System Support
  - Programmed to handle arbitrarily long filenames and messages.
  - GZIP compression on a file by file basis done by the Client program if requested before network transit.
  - Saves and restores POSIX ACLs and Extended Attributes on most OSes if enabled.
  - Access control lists for Consoles that permit restricting user access to only their data.
  - Support for save/restore of files larger than 2GB.
  - Support for 64 bit machines, e.g. amd64, Sparc.
  - Support ANSI and IBM tape labels.
  - Support for Unicode filenames (e.g. Chinese) on Win32 machines
  - Consistent backup of open files on Win32 systems (WinXP, Win2003, and Vista) but not Win2000, using Volume Shadow Copy (VSS).

  - Support for path/filename lengths of up to 64K on Win32 machines (unlimited on Unix/Linux machines).
- Miscellaneous
  - Multi-threaded implementation.
  - A comprehensive and extensible configuration file for each daemon.

13.2 Advantages Over Other Backup Programs

- Since there is a client for each machine, you can backup and restore clients of any type ensuring that all attributes of files are properly saved and restored.
- It is also possible to backup clients without any client software by using NFS or Samba. However, if possible, we recommend running a Client File daemon on each machine to be backed up.
- Bacula handles multi-volume backups.
- A full comprehensive SQL standard database of all files backed up. This permits online viewing of files saved on any particular Volume.
- Automatic pruning of the database (removal of old records) thus simplifying database administration.
- Any SQL database engine can be used making Bacula very flexible. Drivers currently exist for `MySQL`, `PostgreSQL`, and `SQLite` . Note: SQLite is deprecated; it is strongly recommended not to use it.
- The modular but integrated design makes Bacula very scalable.
- Since Bacula uses client file servers, any database or other application can be properly shutdown by Bacula using the native tools of the system, backed up, then restarted (all within a Bacula Job).
- Bacula has a built-in Job scheduler.
- The Volume format is documented and there are simple `C` programs to read/write it.
- Bacula uses well defined (IANA registered) TCP/IP ports – no rpcs, no shared memory.
- Bacula installation and configuration is relatively simple compared to other comparable products.
- According to one user Bacula is as fast as the big major commercial applications.
- According to another user Bacula is four times as fast as another commercial application, probably because that application stores its catalog information in a large number of individual files rather than an SQL database as Bacula does.
- Aside from several GUI administrative interfaces, Bacula has a comprehensive shell administrative interface, which allows the administrator to use tools such as `ssh` to administrate any part of Bacula from anywhere (even from home).
- Bacula has a Rescue CD for Linux systems with the following features:
  - You build it on your own system from scratch with one simple command: `make` – well, then make burn.
  - It uses your kernel
  - It captures your current disk parameters and builds scripts that allow you to automatically repartition a disk and format it to put it back to what you had before.
  - It has a script that will restart your networking (with the right IP address)
  - It has a script to automatically mount your hard disks.
  - It has a full Bacula FD statically linked
  - You can easily add additional data/programs, ... to the disk.

13.3 Current Implementation Restrictions

- It is very unusual to attempt to restore two Jobs that ran simultaneously in a single restore, but if you do, please be aware that unless you had data spooling turned on and the spool file held the full contents of both Jobs during the backup, the restore will not work correctly. In other terms, Bacula cannot restore two jobs in the same restore if the Jobs' data blocks were intermixed on the backup medium. The problem is resolved by simply doing two restores, one for each Job. Normally this can happen only if you manually enter specific JobIds to be restored in a single restore Job.
- Bacula can generally restore any backup made from one client to any other client. However, if the architecture is significantly different (i.e. 32 bit architecture to 64 bit or Win32 to Unix), some restrictions may apply (e.g. Solaris door files do not exist on other Unix/Linux machines; there are reports that Zlib compression written with 64 bit machines does not always read correctly on a 32 bit machine).

13.4 Design Limitations or Restrictions

- Names (resource names, Volume names, and such) defined in Bacula configuration files are limited to a fixed number of characters. Currently the limit is defined as 127 characters. Note, this does not apply to filenames, which may be arbitrarily long.
- Command line input to some of the stand alone tools – e.g. `btape`, `bconsole` is restricted to several hundred characters maximum. Normally, this is not a restriction, except in the case of listing multiple Volume names for programs such as `bscan`. To avoid this command line length restriction, please use a `.bsr` file to specify the Volume names.
- Bacula configuration files for each of the components can be any length. However, the length of an individual line is limited to 500 characters after which it is truncated. If you need lines longer than 500 characters for directives such as ACLs where they permit a list of names are character strings simply specify multiple short lines repeating the directive on each line but with different list values.

13.5 Items to Note

- Bacula's Differential and Incremental *normal* backups are based on time stamps. Consequently, if you move files into an existing directory or move a whole directory into the backup fileset after a Full backup, those files will probably not be backed up by an Incremental save because they will have old dates. This problem is corrected by using Accurate mode backups or by explicitly updating the date/time stamp on all moved files.
- In older versions of Bacula ($\leq$ 3.0.x), if you have over 4 billion file entries stored in your database, the database FileId is likely to overflow. This limitation does not apply to current Bacula versions.
- In non *Accurate* mode, files deleted after a Full save will be included in a restoration. This is typical for most similar backup programs. To avoid this, use Accurate mode backup.

Chapter 14

System Requirements

- **Bacula** has been compiled and runs on Linux, FreeBSD, MacOSX, Solaris, and many other Linux/Unix systems.
- It requires GNU C++ version 2.95 or higher to compile. You can try with other compilers and older versions, but you are on your own. We have successfully compiled and used Bacula using GNU C++ version 4.1.3. Note, in general GNU C++ is a separate package (e.g. RPM) from GNU C, so you need them both loaded. On Red Hat systems, the C++ compiler is part of the **gcc-c++** rpm package.
- There are certain third party packages that Bacula may need. Except for `MySQL` and `PostgreSQL`, they can all be found in the **depkgs** and **depkgs1** releases. However, most current Linux and FreeBSD systems provide these as system packages.
- The minimum versions for each of the databases supported by Bacula are:
  - `MySQL` 5.1
  - `PostgreSQL` 9.0
- If you want a File daemon for Windows, we recommend that you consult the site: www.baculasystems.com, where the binaries are available at a very modest cost. If you are a Bacula Systems customer, the Windows binaries are included with your basic subscription.
- **Bacula** requires a good implementation of pthreads to work. This is not the case on some of the BSD systems.
- The source code has been written with portability in mind and is mostly POSIX compatible. Thus porting to any POSIX compatible operating system should be relatively easy.
- The `GNOME Console` program is developed and tested under GNOME 2.x. GNOME 1.4 is no longer supported.
- The `wxWidgets Console` program is developed and tested with the latest stable ANSI or Unicode version of wxWidgets[1] (2.6.1). It works fine with the Windows and GTK+-2.x version of wxWidgets, and should also work on other platforms supported by wxWidgets.
- The `Tray Monitor` program is developed for GTK+-2.x. It needs GNOME less or equal to 2.2, KDE greater or equal to 3.1 or any window manager supporting the FreeDesktop system tray standard[2].
- If you want to enable command line editing and history, you will need to have `/usr/include/termcap.h` and either the `termcap` or the `ncurses` library loaded (libtermcap-devel or ncurses-devel).
- If you want to use DVD as backup medium, you will need to download the dvd+rw-tools 5.21.4.10.8[3], apply the patch that is in the `patches` directory of the main source tree to

[1] http://www.wxwidgets.org
[2] http://www.freedesktop.org/Standards/systemtray-spec
[3] http://fy.chalmers.se/ appro/linux/DVD+RW/

make these tools compatible with Bacula, then compile and install them. There is also a patch for dvd+rw-tools version 6.1, and we hope that the patch is integrated into a later version. Do not use the dvd+rw-tools provided by your distribution, unless you are sure it contains the patch. dvd+rw-tools without the patch will not work with Bacula. DVD media is not recommended for serious or important backups because of its low reliability.

Chapter 15

Supported Operating Systems

This section applies to the community version only. If you are a Bacula Systems customer, binaries are available for all supported systems. Please see www.baculasystems.com for the current list.

Yes Fully supported

⋆ The are reported to work in many cases and the Community has committed code for them. However they are not directly supported by the Bacula project, as we don't have the hardware.

Table 15.1: Bacula supported operating systems

Operating Systems	Version	Client Daemon	Director Daemon	Storage Deamon
GNU/Linux	All	Yes	Yes	Yes
FreeBSD	⩾ 5.0	Yes	Yes	Yes
Solaris	⩾ 8	Yes	Yes	Yes
MS Windows 32bit	Win98/Me/XP	Yes		
	2K	Yes	⋆	⋆
MS Windows 64bit	XP	Yes	⋆	⋆
	Vista/Win7/Win8	Yes	⋆	⋆
	2000/2003	Yes	⋆	⋆
	2008/2012	Yes	⋆	⋆
	2008/Vista	Yes	⋆	⋆
MacOS X/Darwin		Yes	⋆	⋆
AIX	⩾ 4.3	⋆		
HPUX		⋆		

Important notes

- By GNU/Linux, we mean 32/64bit Gentoo, Red Hat, Fedora, Mandriva, Debian, OpenSuSE, Ubuntu, Kubuntu, ...
- For FreeBSD older than version 5.0, please see some **important** considerations in the **Tape Modes on FreeBSD** section (section 3.3.6 page 36) of Bacula Community Edition

Problems Resolution Guide.

- For MacOSX see for obtaining the packages[1]

See the **Porting** chapter (chapter 11 page 97) of the Bacula Community Edition Developer's manual for information on porting to other systems.

If you have a older Red Hat Linux system running the 2.4.x kernel and you have the directory `/lib/tls` installed on your system (normally by default), bacula will **not** run. This is the new pthreads library and it is defective. You must remove this directory prior to running Bacula, or you can simply change the name to `/lib/tls-broken`) then you must reboot your machine (one of the few times Linux must be rebooted). If you are not able to remove/rename `/lib/tls`, an alternative is to set the environment variable "LD_ASSUME_KERNEL=2.4.19" prior to executing Bacula. For this option, you do not need to reboot, and all programs other than Bacula will continue to use `/lib/tls`. The above mentioned `/lib/tls` problem does not occur with Linux 2.6 kernels.

[1] http://fink.sourceforge.net/

Chapter 16

Supported Tape Drives

Bacula uses standard operating system calls (`read`, `write`, `ioctl`) to interface to tape drives. As a consequence, it relies on having a correctly written OS tape driver. Bacula is known to work perfectly well with SCSI tape drivers on FreeBSD, Linux, Solaris, and Windows machines, and it may work on other *nix machines.

Recently there are many new drives that use IDE, ATAPI, or SATA interfaces rather than SCSI. On Linux the OnStream drive, which uses the OSST driver is one such example, and it is known to work with Bacula. In addition a number of such tape drives (i.e. OS drivers) seem to work on Windows systems. However, non-SCSI tape drives (other than the OnStream) that use `ide-scsi`, `ide-tape`, or other non-scsi drivers do not function correctly with Bacula (or any other demanding tape application) as of today (April 2007). If you have purchased a non-SCSI tape drive for use with Bacula on Linux, there is a good chance that it will not work. We are working with the kernel developers to rectify this situation, but it will not be resolved in the near future.

Generally any modern tape drive (i.e. after 2010) will work out of the box with Bacula using the standard Bacula Device specification in the `bacula-sd.conf` file.

Even if your drive is on the list below, please check the **Tape Testing** section (section 3.2 page 27) of the Bacula Community Edition Problems Resolution Guide for procedures that you can use to verify if your tape drive will work with Bacula. If your drive is in fixed block mode, it may appear to work with Bacula until you attempt to do a restore and Bacula wants to position the tape. You can be sure only by following the procedures suggested above and testing.

It is very difficult to supply a list of supported tape drives, or drives that are known to work with Bacula because of limited feedback (so if you use Bacula on a different drive, please let us know). Based on user feedback, the following drives are known to work with Bacula. A dash in a column means unknown:

Table 16.1: Supported Tape Drives

OS	Manuf.	Media	Model	Capacity
–	ADIC	DLT	Adic Scalar 100 DLT	100GB
–	ADIC	DLT	Adic Fastor 22 DLT	–
FreeBSD 5.4-RELEASE-p1 amd64	Certance	LTO	AdicCertance CL400 LTO Ultrium 2	200GB
–	–	DDS	Compaq DDS 2,3,4	–

Continues on the following page

[]Cont.

OS	Manuf.	Media	Model	Capacity
SuSE 8.1 Pro	Compaq	AIT	Compaq AIT 35 LVD	35/70GB
–	HP	Travan 4	Colorado T4000S	–
–	HP	DLT	HP DLT drives	–
–	HP	LTO	HP LTO Ultrium drives	–
–	IBM	Unknown	3480, 3480XL, 3490, 3490E, 3580 and 3590 drives	–
FreeBSD 4.10 RELEASE	HP	DAT	HP StorageWorks DAT72i	–
–	Overland	LTO	LoaderXpress LTO	–
–	Overland	–	Neo2000	–
–	OnStream	–	OnStream drives (see below)	–
FreeBSD 4.11-Release	Quantum	SDLT	SDLT320	160/320GB
–	Quantum	DLT	DLT-8000	40/80GB
Linux	Seagate	DDS-4	Scorpio 40	20/40GB
FreeBSD 4.9 STABLE	Seagate	DDS-4	STA2401LW	20/40GB
FreeBSD 5.2.1 pthreads patched RELEASE	Seagate	AIT-1	STA1701W	35/70GB
Linux	Sony	DDS-2,3,4	–	4-40GB
Linux	Tandberg	–	Tandbert MLR3	–
FreeBSD	Tandberg	–	Tandberg SLR6	–
Solaris	Tandberg	–	Tandberg SLR75	–

There is a list of supported autochangers in the Supported Autochangers chapter of this document, where you will find other tape drives that work with Bacula.

16.1 Unsupported Tape Drives

Previously OnStream IDE-SCSI tape drives did not work with Bacula. As of Bacula version 1.33 and the osst kernel driver version 0.9.14 or later, they now work. Please see the testing chapter as you must set a fixed block size.

QIC tapes are known to have a number of particularities (fixed block size, and one EOF rather than two to terminate the tape). As a consequence, you will need to take a lot of care in configuring them to make them work correctly with Bacula.

16.2 FreeBSD Users Be Aware!!!

Unless you have patched the pthreads library on FreeBSD 4.11 systems, you will lose data when Bacula spans tapes. This is because the unpatched pthreads library fails to return a warning status to Bacula that the end of the tape is near. This problem is fixed in FreeBSD systems released after 4.11. Please see the **Tape testing** section (section 3.3.6 page 36) of Bacula Community Edition Problems Resolution Guide for **important** information on how to configure your tape drive for compatibility with Bacula.

16.3 Supported Autochangers

For information on supported autochangers, please see the Autochangers Known to Work with Bacula section of the Supported Autochangers chapter of this manual.

16.4 Tape Specifications

If you want to know what tape drive to buy that will work with Bacula, we really cannot tell you. However, we can say that if you are going to buy a drive, you should try to avoid DDS drives. The technology is rather old and DDS tape drives need frequent cleaning. DLT drives are generally much better (newer technology) and do not need frequent cleaning.

Below, you will find a table of DLT and LTO tape specifications that will give you some idea of the capacity and speed of modern tapes. The capacities that are listed are the native tape capacity without compression. All modern drives have hardware compression, and manufacturers often list compressed capacity using a compression ration of 2:1. The actual compression ratio will depend mostly on the data you have to backup, but I find that 1.5:1 is a much more reasonable number (i.e. multiply the value shown in the table by 1.5 to get a rough average of what you will probably see). The transfer rates are rounded to the nearest GB/h. All values are provided by various manufacturers.

The Media Type is what is designated by the manufacturers and you are not required to use (but you may) the same name in your Bacula conf resources.

Table 16.2: DLT & LTO specifications

Media Type	Drive Type	Media Capacity	Transfer Rate
DDS-1	DAT	2 GB	—
DDS-2	DAT	4 GB	—
DDS-3	DAT	12 GB	5.4 GB/hr
Travan 40	Travan	20 GB	—
DDS-4	DAT	20 GB	11 GB/hr
VXA-1	Exabyte	33 GB	11 GB/hr
DAT-72	DAT	36 GB	13 GB/hr
DLT IV	DLT8000	40 GB	22 GB/hr
VXA-2	Exabyte	80 GB	22 GB/hr
Half-high Ultrium 1	LTO 1	100 GB	27 GB/hr

Continues on the following page

[]Cont.

Media Type	Drive Type	Media Capacity	Transfer Rate
Ultrium 1	LTO 1	100 GB	54 GB/hr
Super DLT 1	SDLT 220	110 GB	40 GB/hr
VXA-3	Exabyte	160 GB	43 GB/hr
Super DLT I	SDLT 320	160 GB	58 GB/hr
Ultrium 2	LTO 2	200 GB	108 GB/hr
Super DLT II	SDLT 600	300 GB	127 GB/hr
VXA-4	Exabyte	320 GB	86 GB/hr
Ultrium 3	LTO 3	400 GB	216 GB/hr

Chapter 17

Getting Started with Bacula

If you are like me, you want to get Bacula running immediately to get a feel for it, then later you want to go back and read about all the details. This chapter attempts to accomplish just that: get you going quickly without all the details. If you want to skip the section on Pools, Volumes and Labels, you can always come back to it, but please read to the end of this chapter, and in particular follow the instructions for testing your tape drive.

We assume that you have managed to build and install Bacula, if not, you might want to first look at the System Requirements then at the Compiling and Installing Bacula chapter of this manual.

17.1 Understanding Jobs and Schedules

In order to make Bacula as flexible as possible, the directions given to Bacula are specified in several pieces. The main instruction is the job resource, which defines a job. A backup job generally consists of a FileSet, a Client, a Schedule for one or several levels or times of backups, a Pool, as well as additional instructions. Another way of looking at it is the FileSet is what to backup; the Client is who to backup; the Schedule defines when, and the Pool defines where (i.e. what Volume).

Typically one FileSet/Client combination will have one corresponding job. Most of the directives, such as FileSets, Pools, Schedules, can be mixed and matched among the jobs. So you might have two different Job definitions (resources) backing up different servers using the same Schedule, the same Fileset (backing up the same directories on two machines) and maybe even the same Pools. The Schedule will define what type of backup will run when (e.g. Full on Monday, incremental the rest of the week), and when more than one job uses the same schedule, the job priority determines which actually runs first. If you have a lot of jobs, you might want to use JobDefs, where you can set defaults for the jobs, which can then be changed in the job resource, but this saves rewriting the identical parameters for each job. In addition to the FileSets you want to back up, you should also have a job that backs up your catalog.

Finally, be aware that in addition to the backup jobs there are restore, verify, and admin jobs, which have different requirements.

17.2 Understanding Pools, Volumes and Labels

If you have been using a program such as `tar` to backup your system, Pools, Volumes, and labeling may be a bit confusing at first. A Volume is a single physical tape (or possibly a single file) on which Bacula will write your backup data. Pools group together Volumes so that a backup

is not restricted to the length of a single Volume (tape). Consequently, rather than explicitly naming Volumes in your Job, you specify a Pool, and Bacula will select the next appendable Volume from the Pool and request you to mount it.

Although the basic Pool options are specified in the Director's Pool resource, the **real** Pool is maintained in the Bacula Catalog. It contains information taken from the Pool resource (bacula-dir.conf) as well as information on all the Volumes that have been added to the Pool. Adding Volumes to a Pool is usually done manually with the Console program using the `label` command.

For each Volume, Bacula maintains a fair amount of catalog information such as the first write date/time, the last write date/time, the number of files on the Volume, the number of bytes on the Volume, the number of Mounts, etc.

Before Bacula will read or write a Volume, the physical Volume must have a Bacula software label so that Bacula can be sure the correct Volume is mounted. This is usually done using the `label` command in the Console program.

The steps for creating a Pool, adding Volumes to it, and writing software labels to the Volumes, may seem tedious at first, but in fact, they are quite simple to do, and they allow you to use multiple Volumes (rather than being limited to the size of a single tape). Pools also give you significant flexibility in your backup process. For example, you can have a "Daily" Pool of Volumes for Incremental backups and a "Weekly" Pool of Volumes for Full backups. By specifying the appropriate Pool in the daily and weekly backup Jobs, you thereby insure that no daily Job ever writes to a Volume in the Weekly Pool and vice versa, and Bacula will tell you what tape is needed and when.

For more on Pools, see the Pool Resource section of the Director Configuration chapter, or simply read on, and we will come back to this subject later.

17.3 Setting Up Bacula Configuration Files

Normally, if you are using the Bacula Enterprise version, you will install it from packages (.deb, .rpm, .pkg, .dmg, ...), in which case, default configuration files will already be setup in `/opt/bacula/etc`. However, you will need to reconfigure these files to correspond to your production environment including adding definitions of additional Client machines to be backed up.

When initially setting up Bacula you will need to invest a bit of time in modifying the default configuration files to suit your environment. This may entail starting and stopping Bacula a number of times until you get everything right. Please do not despair. Once you have created your production configuration files, you will rarely need to change them nor will you stop and start Bacula very often. Most of the work will simply be in monitoring that there is sufficient disk space or tapes available and that no jobs are failing.

17.3.1 Configuring the Console Program

The Console program is used by the administrator to interact with the Director and to manually start/stop Jobs or to obtain Job status information.

The Console configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the `./configure` command and by default is named `bconsole.conf`.

The same applies to the wxWidgets console, which is build with the **--enable-bwx-console** option, and the name of the default configuration file is, in this case, `bwx-console.conf`.

Normally, for first time users, no change is needed to these files. Reasonable defaults are set.

Further details are in the Console configuration chapter.

17.3.2 Configuring the File daemon

The File daemon is a program that runs on each (Client) machine. At the request of the Director, finds the files to be backed up and sends them (their data) to the Storage daemon.

The File daemon configuration file is found in the directory specified on the --**sysconfdir** option that you specified on the `./configure` command. By default, the File daemon's configuration file is named `bacula-fd.conf`. Normally, for first time users, no change is needed to this file. Reasonable defaults are set. However, if you are going to back up more than one machine, you will need to install the File daemon with a unique configuration file on each machine to be backed up. The information about each File daemon must appear in the Director's configuration file.

Further details are in the File daemon configuration chapter.

17.3.3 Configuring the Director

The Director is the central control program for all the other daemons. It schedules and monitors all jobs to be backed up.

The Director configuration file is found in the directory specified on the --**sysconfdir** option that you specified on the `./configure` command. Normally the Director's configuration file is named `bacula-dir.conf`.

In general, the only change you must make is modify the FileSet resource so that the **Include** configuration directive contains at least one line with a valid name of a directory (or file) to be saved.

If you do not have a DLT tape drive, you will probably want to edit the Storage resource to contain names that are more representative of your actual storage device. You can always use the existing names as you are free to arbitrarily assign them, but they must agree with the corresponding names in the Storage daemon's configuration file.

You may also want to change the email address for notification from the default **root** to your email address.

Finally, if you have multiple systems to be backed up, you will need a separate File daemon or Client specification for each system, specifying its name, address, and password. We have found that giving your daemons the same name as your system but post fixed with -**fd** helps a lot in debugging. That is, if your system name is **foobaz**, you would give the File daemon the name **foobaz-fd**. For the Director, you should use **foobaz-dir**, and for the storage daemon, you might use **foobaz-sd**. Each of your Bacula components **must** have a unique name. If you make them all the same, aside from the fact that you will not know what daemon is sending what message, if they share the same working directory, the daemons temporary file names will not be unique, and you will get many strange failures.

More information is in the Director configuration chapter.

17.3.4 Configuring the Storage daemon

The Storage daemon is responsible, at the Director's request, for accepting data from a File daemon and placing it on Storage media, or in the case of a restore request, to find the data and send it to the File daemon.

The Storage daemon's configuration file is found in the directory specified on the --**sysconfdir**

option that you specified on the `./configure` command. By default, the Storage daemon's file is named `bacula-sd.conf`. Edit this file to contain the correct Archive device names for any tape devices that you have. If the configuration process properly detected your system, they will already be correctly set. These Storage resource name and Media Type must be the same as the corresponding ones in the Director's configuration file `bacula-dir.conf`. If you want to backup to a file instead of a tape, the Archive device must point to a directory in which the Volumes will be created as files when you label the Volume.

Further information is in the Storage daemon configuration chapter.

17.3.5 Configuring the Monitor Program

The Monitor program is typically an icon in the system tray. However, once the icon is expanded into a full window, the administrator or user can obtain status information about the Director or the backup status on the local workstation or any other Bacula daemon that is configured.

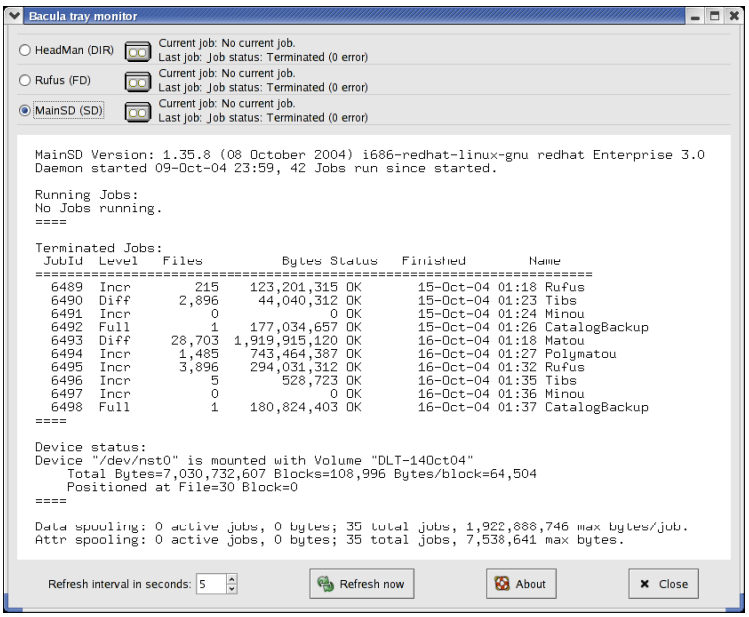


Figure 17.1: Bacula Tray Monitor

The image shows a tray-monitor configured for three daemons. By clicking on the radio buttons in the upper left corner of the image, you can see the status for each of the daemons. The image shows the status for the Storage daemon (MainSD) that is currently selected.

The Monitor configuration file is found in the directory specified on the `--sysconfdir` option that you specified on the `./configure` command and by default is named `tray-monitor.conf`. Normally, for first time users, you just need to change the permission of this file to allow non-root users to run the Monitor, as this application must run as the same user as the graphical environment (don't forget to allow non-root users to execute `bacula-tray-monitor`). This is not a security problem as long as you use the default settings.

More information is in the Monitor configuration chapter.

17.4 Testing your Configuration Files

You can test if your configuration file is syntactically correct by running the appropriate daemon with the **-t** option. The daemon will process the configuration file and print any error messages

then terminate. For example, assuming you have installed your binaries and configuration files in the same directory.

```
cd <installation-directory>
./bacula-dir -t -c bacula-dir.conf
./bacula-fd -t -c bacula-fd.conf
./bacula-sd -t -c bacula-sd.conf
./bconsole -t -c bconsole.conf
./bwx-console -t -c bwx-console.conf
./bat -t -c bat.conf
su <normal user> -c "./bacula-tray-monitor -t -c tray-monitor.conf"
```

will test the configuration files of each of the main programs. If the configuration file is OK, the program will terminate without printing anything. Please note that, depending on the configure options you choose, some, or even all, of the three last commands will not be available on your system. If you have installed the binaries in traditional Unix locations rather than a single file, you will need to modify the above commands appropriately (no ./ in front of the command name, and a path in front of the conf file name).

17.5 Testing Compatibility with Your Tape Drive

Before spending a lot of time on Bacula only to find that it doesn't work with your tape drive, please read the **Testing Your Tape Drive** chapter of this manual. If you have a modern standard SCSI tape drive on a Linux or Solaris, most likely it will work, but better test than be sorry. For FreeBSD (and probably other xBSD flavors), reading the above mentioned tape testing chapter is a must. Also, for FreeBSD, please see The FreeBSD Diary[1] for a detailed description on how to make Bacula work on your system. In addition, users of FreeBSD prior to 4.9-STABLE dated Mon Dec 29 15:18:01 2003 UTC who plan to use tape devices, please see the file `platforms/freebsd/pthreads-fix.txt` in the main Bacula directory concerning important information concerning compatibility of Bacula and your system.

17.6 Get Rid of the /lib/tls Directory

This section only applies to really old 2.4.x kernel version, which we hope you are no longer running.

The new pthreads library `/lib/tls` installed by default on recent Red Hat systems running Linux kernel 2.4.x is defective. You must remove it or rename it, then reboot your system before running Bacula otherwise after a week or so of running, Bacula will either block for long periods or deadlock entirely. You may want to use the loader environment variable override rather than removing /lib/tls. Please see Supported Operating Systems for more information on this problem.

This problem does not occur on systems running Linux 2.6.x kernels.

17.7 Running Bacula

Probably the most important part of running Bacula is being able to restore files. If you haven't tried recovering files at least once, when you actually have to do it, you will be under a lot more pressure, and prone to make errors, than if you had already tried it once.

[1] http://www.freebsddiary.org/bacula.php

To get a good idea how to use Bacula in a short time, we **strongly** recommend that you follow the example in the Running Bacula Chapter of this manual where you will get detailed instructions on how to run Bacula.

17.8 Log Rotation

If you use the default `bacula-dir.conf` or some variation of it, you will note that it logs all the Bacula output to a file. To avoid that this file grows without limit, we recommend that you copy the file `logrotate` from the `scripts/logrotate` to `/etc/logrotate.d/bacula`. This will cause the log file to be rotated once a month and kept for a maximum of five months. You may want to edit this file to change the default log rotation preferences.

17.9 Log Watch

Some systems such as Red Hat and Fedora run the logwatch program every night, which does an analysis of your log file and sends an email report. If you wish to include the output from your Bacula jobs in that report, please look in the `scripts/logwatch` directory. The `README` file in that directory gives a brief explanation on how to install it and what kind of output to expect.

17.10 Disaster Recovery

If you intend to use Bacula as a disaster recovery tool rather than simply a program to restore lost or damaged files, you will want to read the Disaster Recovery Using Bacula Chapter of this manual.

In any case, you are strongly urged to carefully test restoring some files that you have saved rather than wait until disaster strikes. This way, you will be prepared.

Chapter 18

Installing Bacula

18.1 Binary Release Packages

The project releases binary .rpm and .deb packages for popular Linux distributions. These are probably the easiest way to install a new version of Bacula. Please see the Bacula Binary Package Download page on www.bacula.org for downloading these pre-prepared packages.

18.2 Building Bacula from Source

In general, you will need the Bacula source release, and if you want to run a Windows client, you will need the Bacula Windows binary release. However, Bacula needs certain third party packages (such as **MySQL** or **PostgreSQL** to build and run properly depending on the options you specify. Normally, **MySQL** and **PostgreSQL** are packages that can be installed on your distribution. However, if you do not have them, to simplify your task, we have combined a number of these packages into three **depkgs** releases (Dependency Packages). This can vastly simplify your life by providing you with all the necessary packages rather than requiring you to find them on the Web, load them, and install them.

18.3 Source Release Files

The source code has been broken into five separate tar files each corresponding to a different module in the Bacula git repository. The released files are:

bacula-11.0.0.tar.gz This is the primary source code release for Bacula. On each release the version number (11.0.0) will be updated.

bacula-docs-11.0.0.tar.bz2 This file contains a copy of the docs directory with the documents prebuild. English HTML directory, single HTML file, and pdf file. The French, German, Spanish translations are not built, but can be obtained from the git repository. Note that this file is big, it used .bz2 compression which reduces the size.

bacula-gui-11.0.0.tar.gz This file contains the non-core GUI programs. Currently, it contains Baculum, a PHP program for producing management viewing of your Bacula job status in a browser.

bacula-regress-11.0.0.tar.gz This file contains the regression tests that developers and testers use to test Bacula. Community users can help a lot by running these regression tests on various different Unix/Linux distributions.

bacula-sigs-11.0.0.tar.gz This tar file contains the hash coded signature files that are signed by the Bacula project. These signature files are signed by the Bacula project private key, and can be used in conjunction with the Bacula project public key found on the www.bacula.org web site to allow you to verify that the source files that you downloaded are genuine and have no errors.

bacula-win32-11.0.0.exe This file is the 32 bit Windows installer for installing the Windows client (File daemon) on a Windows machine. This client will also run on 64 bit Windows machines, but VSS support is not available if you are running a 64 bit version of Windows. This installer installs only the FD, the Director and Storage daemon are not included.

bacula-win64-11.0.0.exe This file is the 64 bit Windows installer for installing the Windows client (File daemon) on a Windows machine. This client will only run on 64 bit Windows OS machines. It will not run on 32 bit machines or 32 bit Windows OSes. The Bacula Win64 release is necessary for Volume Shadow Copy (VSS) to work on Win64 OSes. This installer installs only the FD and the SD (disk only. The Windows Director daemon is not included as it is not currently supported.

18.4 Upgrading Bacula

If you are upgrading from one Bacula version to another, you should first carefully read the ReleaseNotes of all major versions between your current version and the version to which you are upgrading. In many upgrades, especially for minor patch upgrades (e.g. between 3.0.0 and 3.0.1) there will be no database upgrade, and hence the process is rather simple.

With version 3.0.0 and later, you **must** ensure that on any one machine that all components of Bacula are running on exactly the same version. Prior to version 3.0.0, it was possible to run a lower level FD with a newer Director and SD. This is no longer the case.

As always, we attempt to support older File daemons. This avoids the need to do a simultaneous upgrade of many machines. For exactly what older versions of the FD are supported, please see the ReleaseNotes for the new version. In any case, you must always upgrade both the Director and the Storage daemon at the same time, and you must also upgrade any File daemon that is running on the same machine as a Director or a Storage daemon (see the prior paragraph).

If the Bacula catalog database has been upgraded (as it is almost every major release), you will either need to reinitialize your database starting from scratch (not normally a good idea), or save an ASCII copy of your database, then proceed to upgrade it. If you are upgrading two major versions (e.g. 1.36 to 2.0) then life will be more complicated because you must do two database upgrades. See below for more on this.

Upgrading the catalog is normally done after Bacula is build and installed by:

```
cd <installed-scripts-dir> (default /etc/bacula)
./update_bacula_tables
```

This update script can also be find in the Bacula source src/cats directory.

If you are upgrading from a very old version of Bacula, there may be several catalog database updates needed to go from your existing catalog format to the new one. The **./update_bacula_tables** script will automatically do the updates for you one at a time.

If you are upgrading from one major version to another, you will need to replace all your components at the same time as generally the inter-daemon protocol will change. However, within any particular release (e.g. version 1.32.x) unless there is an oversight or bug, the daemon protocol will not change. If this is confusing, simply read the ReleaseNotes very carefully as they will note if all daemons must be upgraded at the same time.

Finally, please note that in general it is not necessary or desirable to do a **make uninstall** before doing an upgrade providing you are careful not to change the installation directories. In fact, if you do so, you will most likely delete all your conf files, which could be disastrous.

The normal procedure depends on whether you previously installed binary packages, in which case, you will use the update option to install the newer packages. If you installed from souce please follow the following steps:

```
./configure (your options)
make
make install
```

In general none of your existing .conf or .sql files will be overwritten, and you must do both the **make** and **make install** commands, a **make install** without the preceding **make** will not work.

We recommend the following base options for the ./configure that you run to configure how the make process will install Bacula.

```
#!/bin/sh
# This is the recommended configure script for Bacula
PREFIX=/opt/bacula
CFLAGS="-g -O2 -Wall" \
  ./configure \
    --sbindir=${PREFIX}/bin \
    --sysconfdir=${PREFIX}/etc \
    --docdir=${PREFIX}/html \
    --htmldir=${PREFIX}/html \
    --with-working-dir=${PREFIX}/working \
    --with-pid-dir=${PREFIX}/working \
    --with-scriptdir=${PREFIX}/scripts \
    --with-plugindir=${PREFIX}/plugins \
    --libdir=${PREFIX}/lib \
    --enable-smartalloc \
    --enable-conio \
    --enable-bat \
    --with-postgresql \
    --with-dump-email=email@example.com \
    --with-job-email=email@example.com \
    --with-smtp-host=mail.example.com \
    --with-baseport=9101

exit 0
```

Note: you should adjust the email addresses to correspond to the address to which you would like Bacula to send you mail. Also, you must have the correct Qt5 libraries pre-loaded. If not, remove the line "–enable-bat".

18.5 Releases Numbering

Every Bacula release whether beta or production has a different number as well as the date of the release build. The numbering system follows traditional Open Source conventions in that it is of the form.

```
major.minor.release
```

For example:

```
1.38.11
```

where each component (major, minor, patch) is a number. The major number is currently 1 and normally does not change very frequently. The minor number starts at 0 and increases each for each production release by 2 (i.e. it is always an even number for a production release), and the patch number is starts at zero each time the minor number changes. The patch number is increased each time a bug fix (or fixes) is released to production.

So, as of this date (05 May 2019), the current production Bacula release is version 11.0.3. If there are bug fixes, the next release will be 11.0.2 (i.e. the patch number has increased by one).

For all patch releases where the minor version number does not change, the database and all the daemons will be compatible. That means that you can safely run a 11.0.2 Director with a 11.0.0 Client. Of course, in this case, the Director may have bugs that are not fixed. Generally, within a minor release (some minor releases are not so minor), all patch numbers are officially released to production. This means that while the current Bacula version is 11.0.3, versions 11.0.0, 11.0.1, and 11.0.2 have been previously released.

When the minor number is odd, it indicates that the package is under development and thus may not be stable. For example, while the current production release of Bacula is currently 11.0.2, the current development version is 9.5.x. All patch versions of the development code are available in the SVN (source repository). However, not all patch versions of the development code (odd minor version) are officially released. When they are released, they are released as beta versions (see below for a definition of what beta means for Bacula releases).

In general when the minor number increases from one production release to the next (i.e. 9.2.5 to 11.0.0), the catalog database must be upgraded, the Director and Storage daemon must always be on the same minor release number, and often (not always), the Clients must also be on the same minor release. As often as possible, we attempt to make new releases that are downwards compatible with prior clients, but this is not always possible. You must check the release notes. In general, you will have fewer problems if you always run all the components on the same minor version number (i.e. all either 9.2.x or 11.0.x but not mixed).

Beta Releases

Towards the end of the development cycle, which typically runs one year from a major release to another, there will be several beta releases of the development code prior to a production release. As noted above, beta versions always have odd minor version numbers (e.g 9.1.x or 9.5.x). The purpose of the beta releases is to allow early adopter users to test the new code. Beta releases are made with the following considerations:

- The code passes the regression testing on FreeBSD, Linux, and Solaris machines.
- There are no known major bugs, or on the rare occasion that there are, they will be documented or already in the bugs database.
- Some of the new code/features may not yet be tested.
- Bugs are expected to be found, especially in the new code before the final production release.
- The code will have been run in production in at least one small site (mine).
- The Win32 client will have been run in production at least one night at that small site.
- The documentation in the manual is unlikely to be complete especially for the new features, and the Release Notes may not be fully organized.
- Beta code is not generally recommended for everyone, but rather for early adopters.

18.6 Dependency Packages

As discussed above, we have combined a number of third party packages that Bacula might need into the **depkgs** release. You can, of course, get the latest packages from the original authors or from your operating system supplier. The locations of where we obtained the packages are in the README file in each package. However, be aware that the packages in the depkgs files have been tested by us for compatibility with Bacula.

Typically, a dependency package will be named **depkgs-ddMMMyy.tar.gz** where **dd** is the day we release it, **MMM** is the abbreviated month (e.g. Jan), and **yy** is the year. An actual example is: **depkgs-18Dec.tar.gz**. To install and build this package (if needed), you do the following:

1. Create a **bacula** directory, into which you will place both the Bacula source as well as the dependency package.
2. Detar the **depkgs** into the **bacula** directory.
3. cd bacula/depkgs
4. make

Although the exact composition of the dependency packages may change from time to time, the current makeup is the following:

3rd Party Package	depkgs	depkgs-qt
mtx	X	
qt4		X

Note, some of these packages are quite large, so that building them can be a bit time consuming. The above instructions will build all the packages contained in the directory. However, when building Bacula, it will take only those pieces that it actually needs.

Alternatively, you can make just the packages that are needed. For example,

```
cd bacula/depkgs
make qt4
```

will configure and build only the QT4 package.

You should build the packages that you will require in **depkgs** a prior to configuring and building Bacula, since Bacula will need them during the build process.

Note, the **depkgs-qt** package is required for building bat, because bat is currently built with Qt version 4.3.4. It can be built with other Qt versions, but that almost always creates problems or introduces instabilities.

You can build the depkgs-qt with the following:

```
cd bacula
tar xfvz depkgs-qt-28Jul09.tar.gz
cd depkgs-qt
make qt4
source qt4-path
```

Doing the **source qt4-path** defines the following environment variables:

```
QTDIR
QTLIB
QTINC
```

Each one should point to a specific location in the depkgs-qt package that you loaded. It also puts the depkgs-qt/qt4/bin directory on your path before all other directories. This ensures that the bat build will use your Qt 4.3.4 library rather than any that might be on your system.

Before running your Bacula build, please make sure that **qmake-qt4** is not on your path. If it is please rename it. If you don't do this, Bacula will attempt to build with any Qt5 package installed on your system rather than the one you just built. If you logoff and log back in, you must re-source the depkgs-qt/qt4-patch file before attempting to rebuild the bat part of Bacula.

For more information on the **depkgs-qt** package, please read the INSTALL file in the main directory of that package. If you are going to build Qt5 using **depkgs-qt**, you must source the **qt5-paths** file included in the package prior to building Bacula. Please read the INSTALL file for more details.

You might find it worthwhile to build **mtx** because the **tapeinfo** program that comes with it can often provide you with valuable information about your SCSI tape drive (e.g. compression, min/max block sizes, ...). Note, most distros provide **mtx** as part of their release.

The **depkgs1** package is depreciated and previously contained readline, which should be available on all operating systems.

The **depkgs-win32** package is deprecated and no longer used in Bacula version 1.39.x and later. It was previously used to build the native Win32 client program, but this program is now built on Linux systems using cross-compiling. All the tools and third party libraries are automatically downloaded by executing the appropriate scripts. See src/win32/README.mingw32 for more details.

18.7 Supported Operating Systems

Please see the Supported Operating Systems section of the QuickStart chapter of this manual.

18.8 Building Bacula from Source

The basic installation is rather simple.

1. Install and build any **depkgs** as noted above. This should be unnecessary on most modern Operating Systems.
2. Configure and install MySQL or PostgreSQL (if desired). Installing and Configuring MySQL Phase I or Installing and Configuring PostgreSQL Phase I. If you are installing from rpms, and are using MySQL, please be sure to install **mysql-devel**, so that the MySQL header files are available while compiling Bacula. In addition, the MySQL client library **mysqlclient** requires the gzip compression library **libz.a** or **libz.so**. If you are using rpm packages, these libraries are in the **libz-devel** package. On Debian systems, you will need to load the **zlib1g-dev** package. If you are not using rpms or debs, you will need to find the appropriate package for your system.

   Note, if you already have a running MySQL or PostgreSQL on your system, you can skip this phase provided that you have built the thread safe libraries. And you have already installed the additional rpms noted above.
3. Detar the Bacula source code preferably into the **bacula** directory discussed above.
4. **cd** to the directory containing the source code.
5. ./configure (with appropriate options as described below). Any path names you specify as options on the ./configure command line must be absolute paths and not relative.

6. Check the output of ./configure very carefully, especially the Install binaries and Install config directories. If they are not correct, please rerun ./configure until they are. The output from ./configure is stored in **config.out** and can be re-displayed at any time without rerunning the ./configure by doing **cat config.out**.

7. If after running ./configure once, you decide to change options and re-run it, that is perfectly fine, but before re-running it, you should run:

   ```
   make distclean
   ```

   so that you are sure to start from scratch and not have a mixture of the two options. This is because ./configure caches much of the information. The **make distclean** is also critical if you move the source directory from one machine to another. If the **make distclean** fails, just ignore it and continue on.

8. make If you get errors while linking in the Storage daemon directory (src/stored), it is probably because you have not loaded the static libraries on your system. I noticed this problem on a Solaris system. To correct it, make sure that you have not added - **-enable-static-tools** to the **./configure** command.

   If you skip this step (**make**) and proceed immediately to the **make install** you are making two serious errors: 1. your install will fail because Bacula requires a **make** before a **make install**. 2. you are depriving yourself of the chance to make sure there are no errors before beginning to write files to your system directories.

9. make install Please be sure you have done a **make** before entering this command, and that everything has properly compiled and linked without errors.

10. If you are new to Bacula, we **strongly** recommend that you skip the next step and use the default configuration files, then run the example program in the next chapter, then come back and modify your configuration files to suit your particular needs.

11. Customize the configuration files for each of the three daemons (Directory, File, Storage) and for the Console program. For the details of how to do this, please see Setting Up Bacula Configuration Files in the Configuration chapter of this manual. We recommend that you start by modifying the default configuration files supplied, making the minimum changes necessary. Complete customization can be done after you have Bacula up and running. Please take care when modifying passwords, which were randomly generated, and the **Name**s as the passwords and names must agree between the configuration files for security reasons.

12. Create the Bacula MySQL database and tables (if using MySQL) Installing and Configuring MySQL Phase II or create the Bacula PostgreSQL database and tables Configuring PostgreSQL II or alternatively if you are using

13. Start Bacula (**./bacula start**) Note. the next chapter shows you how to do this in detail.

14. Interface with Bacula using the Console program

15. For the previous two items, please follow the instructions in the Running Bacula chapter of this manual, where you will run a simple backup and do a restore. Do this before you make heavy modifications to the configuration files so that you are sure that Bacula works and are familiar with it. After that changing the conf files will be easier.

16. If after installing Bacula, you decide to "move it", that is to install it in a different set of directories, proceed as follows:

    ```
    make uninstall
    make distclean
    ./configure (your-new-options)
    make
    make install
    ```

If all goes well, the **./configure** will correctly determine which operating system you are running and configure the source code appropriately. Currently, FreeBSD, Linux (Red Hat), and Solaris are supported. The Bacula client (File daemon) is reported to work with MacOS X 10.3 is if readline support is not enabled (default) when building the client.

If you install Bacula on more than one system, and they are identical, you can simply transfer the source tree to that other system and do a "make install". However, if there are differences in the libraries or OS versions, or you wish to install on a different OS, you should start from the original compress tar file. If you do transfer the source tree, and you have previously done a ./configure command, you MUST do:

```
make distclean
```

prior to doing your new ./configure. This is because the GNU autoconf tools cache the configuration, and if you re-use a configuration for a Linux machine on a Solaris, you can be sure your build will fail. To avoid this, as mentioned above, either start from the tar file, or do a "make distclean".

In general, you will probably want to supply a more complicated **configure** statement to ensure that the modules you want are built and that everything is placed into the correct directories.

For example, on Fedora, Red Hat, or SuSE one could use the following:

```
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=/opt/bacula/bin \
    --sysconfdir=/opt/bacula/etc \
    --with-pid-dir=/var/run \
    --with-subsys-dir=/var/run \
    --with-mysql \
    --with-working-dir=/opt/bacula/working \
    --with-dump-email=$USER
```

The advantage of using the above configuration to start is that everything will be put into a single directory, which you can later delete once you have run the examples in the next chapter and learned how Bacula works. In addition, the above can be installed and run as non-root.

For the developer's convenience, I have added a **defaultconfig** script to the **examples** directory. This script contains the statements that you would normally use, and each developer/user may modify them to suit his needs. You should find additional useful examples in this directory as well.

The **--enable-conio** or **--enable-readline** options are useful because they provide a command line history, editing capability for the Console program and tab completion on various option. If you have included either option in the build, either the **termcap** or the **ncurses** package will be needed to link. On most systems, including Red Hat and SuSE, you should include the ncurses package. If Bacula's configure process finds the ncurses libraries, it will use those rather than the termcap library. On some systems, such as SuSE, the termcap library is not in the standard library directory. As a consequence, the option may be disabled or you may get an error message such as:

```
/usr/lib/gcc-lib/i586-suse-linux/3.3.1/.../ld:
cannot find -ltermcap
collect2: ld returned 1 exit status
```

while building the Bacula Console. In that case, you will need to set the **LDFLAGS** environment variable prior to building.

```
export LDFLAGS="-L/usr/lib/termcap"
```

The same library requirements apply if you wish to use the readline subroutines for command line editing, history and tab completion or if you are using a MySQL library that requires encryption. If you need encryption, you can either export the appropriate additional library options as shown above or, alternatively, you can include them directly on the ./configure line as in:

```
LDFLAGS="-lssl -lcyrpto" \
   ./configure <your-options>
```

On some systems such as Mandriva, readline tends to gobble up prompts, which makes it totally useless. If this happens to you, use the disable option, or if you are using version 1.33 and above try using **--enable-conio** to use a built-in readline replacement. You will still need either the termcap or the ncurses library, but it is unlikely that the **conio** package will gobble up prompts.

readline is no longer supported after version 1.34. The code within Bacula remains, so it should be usable, and if users submit patches for it, we will be happy to apply them. However, due to the fact that each version of readline seems to be incompatible with previous versions, and that there are significant differences between systems, we can no longer afford to support it.

18.9 What Database to Use?

If you wish to use MySQL as the Bacula catalog, please see the Installing and Configuring MySQL chapter of this manual. You will need to install MySQL prior to continuing with the configuration of Bacula. MySQL is a high quality database that is very efficient and is suitable for small and medium sized installation (up to 2,000,000 files per job).

If you wish to use PostgreSQL as the Bacula catalog, please see the Installing and Configuring PostgreSQL chapter of this manual. You will need to install PostgreSQL prior to continuing with the configuration of Bacula. PostgreSQL is very similar to MySQL, though it tends to be slightly more SQL92 compliant and has more advanced features such as DDL support in transactions. It requires a certain knowledge to install and maintain. PostgreSQL is suitable for any sized installation (some sites have much more than 1 billion objects in the Catalog). Bacula uses many optimized PostgreSQL functions, and can run more than 10 time faster on jobs having millions of files than MySQL (Specially in during restore, accurate mode, bvfs queries and when the database server is not on the same host than the Director).

18.10 Quick Start

There are a number of options and important considerations given below that you can skip for the moment if you have not had any problems building Bacula with a simplified configuration as shown above.

If the ./configure process is unable to find specific libraries (e.g. libintl, you should ensure that the appropriate package is installed on your system. Alternatively, if the package is installed in a non-standard location (as far as Bacula is concerned), then there is generally an option listed below (or listed with "./configure - -help" that will permit you to specify the directory that should be searched. In other cases, there are options that will permit you to disable to feature (e.g. - -disable-nls).

If you want to dive right into it, we recommend you skip to the next chapter, and run the example program. It will teach you a lot about Bacula and as an example can be installed into a single directory (for easy removal) and run as non-root. If you have any problems or when you want to do a real installation, come back to this chapter and read the details presented below.

18.11 Configure Options

The following command line options are available for **configure** to customize your installation.

-prefix=<patch> This option is meant to allow you to direct where the architecture independent files should be placed. However, we find this a somewhat vague concept, and so we have not implemented this option other than to use any explicit prefix that you may define. If you do not explicitly specify a prefix, Bacula's configure routine will not use the default value that ./configure –help prints. As a consequence, we suggest that you avoid it. We have provided options that allow you to explicitly specify the directories for each of the major categories of installation files.

-sbindir=<binary-path> Defines where the Bacula binary (executable) files will be placed during a **make install** command.

-sysconfdir=<config-path> Defines where the Bacula configuration files should be placed during a **make install** command. Note, for security reasons, this directory should be unique to Bacula and not read/writable by any other user/group than Bacula is running under.

Please note that Bacula attempts to make the configuration directory secure by doing an: chmod 770 <sysconfdir> So, if you make sysconfdir point to a global directory such

as /usr/local/etc, the modes for that directory will be restricted, and this may affect other programs that are installed there. We strongly recommend that you set **sysconfdir** to **/opt/bacula/etc** so that the directory for all Bacula configuration files is unique to Bacula and used only by Bacula.

-mandir=<path> Note, as of Bacula version 1.39.14, the meaning of any path specified on this option is change from prior versions. It now specifies the top level man directory. Previously the mandir specified the full path to where you wanted the man files installed. The man files will be installed in gzip'ed format under mandir/man1 and mandir/man8 as appropriate. For the install to succeed you must have **gzip** installed on your system.

By default, Bacula will install the Unix man pages in /usr/share/man/man1 and /usr/share/man/man8. If you wish the man page to be installed in a different location, use this option to specify the path. Note, the main HTML and PDF Bacula documents are in a separate tar file that is not part of the source distribution.

-datadir=<path> If you translate Bacula or parts of Bacula into a different language you may specify the location of the po files using the **-datadir** option. You must manually install any po files as Bacula does not (yet) automatically do so.

-disable-ipv6

-enable-smartalloc This enables the inclusion of the Smartalloc orphaned buffer detection code. This option is highly recommended. Because we never build without this option, you may experience problems if it is not enabled. In this case, simply re-enable the option. We strongly recommend keeping this option enabled as it helps detect memory leaks. This configuration parameter is used while building Bacula

-enable-bat If you have Qt5 installed on your computer including the libqt4 and libqt4-devel (libqt4-dev on Debian) libraries, and you want to use the Bacula Administration Tool (bat) GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/qt-console** directory. The build with enable-bat will work only with a full Bacula build (i.e. it will not work with a client-only build).

Qt5 is available on OpenSUSE, CentOS , Fedora, and Debian. If it is not available on your system, you can download the **depkgs-qt** package from the Bacula Source Forge download area and build it. See the INSTALL file in that package for more details. In particular to use the Qt5 built by **depkgs-qt** you **must** source the file **qt5-paths**.

-enable-batch-insert This option enables batch inserts of the attribute records (default) in the catalog database, which is much faster (10 times or more) than without this option for large numbers of files. However, this option will automatically be disabled if your SQL

libraries are not thread safe. If you find that batch mode is not enabled on your Bacula installation, then your database most likely does not support threads.

On most systems, MySQL and PostgreSQLare thread safe.

To verify that your PostgreSQL is thread safe, you can try this (change the path to point to your particular installed libpq.a; these commands were issued on FreeBSD 6.2):

```
$ nm /usr/local/lib/libpq.a | grep PQputCopyData
00001b08 T PQputCopyData
$ nm /usr/local/lib/libpq.a | grep mutex
         U pthread_mutex_lock
         U pthread_mutex_unlock
         U pthread_mutex_init
         U pthread_mutex_lock
         U pthread_mutex_unlock
```

The above example shows a libpq that contains the required function PQputCopyData and is thread enabled (i.e. the pthread_mutex* entries). If you do not see PQputCopyData, your version of PostgreSQL is too old to allow batch insert. If you do not see the mutex entries, then thread support has not been enabled. Our tests indicate you usually need to change the configuration options and recompile/reinstall the PostgreSQL client software to get thread support.

Bacula always links to the thread safe MySQL libraries.

Running with Batch Insert turned on is recommended because it can significantly improve attribute insertion times. However, it does put a significantly larger part of the work on your SQL engine, so you may need to pay more attention to tuning it. In particular, Batch Insert can require large temporary table space, and consequently, the default location (often /tmp) may run out of space causing errors. For MySQL, the location is set in my.conf with "tmpdir". You may also want to increase the memory available to your SQL engine to further improve performance during Batch Inserts.

-enable-bwx-console If you have wxWidgets installed on your computer and you want to use the wxWidgets GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/wx-console** directory. This could also be useful to users who want a GUI Console and don't want to install QT, as wxWidgets can work with GTK+, Motif or even X11 libraries.

-enable-static-tools This option causes the linker to link the Storage daemon utility tools (**bls**, **bextract**, and **bscan**) statically. This permits using them without having the shared libraries loaded. If you have problems linking in the **src/stored** directory, make sure you have not enabled this option, or explicitly disable static linking by adding **--disable-static-tools**.

-enable-static-fd This option causes the make process to build a **static-bacula-fd** in addition to the standard File daemon. This static version will include statically linked libraries and is required for the Bare Metal recovery. This option is largely superseded by using **make static-bacula-fd** from with in the **src/filed** directory. Also, the **--enable-client-only** option described below is useful for just building a client so that all the other parts of the program are not compiled.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **-openssl** or other options on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

-enable-static-sd This option causes the make process to build a **static-bacula-sd** in addition to the standard Storage daemon. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **-openssl** or othyer options on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

-enable-static-dir This option causes the make process to build a **static-bacula-dir** in addition to the standard Director. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **-openssl** or other options on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

-enable-static-cons This option causes the make process to build a **static-console** in addition to the standard console. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **-openssl** or other options on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

-enable-client-only This option causes the make process to build only the File daemon and the libraries that it needs. None of the other daemons, storage tools, nor the console will be built. Likewise a **make install** will then only install the File daemon. To cause all daemons to be built, you will need to do a configuration without this option. This option greatly facilitates building a Client on a client only machine.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **-openssl** or other options on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

-enable-build-dird This option causes the make process to build the Director and the Director's tools. By default, this option is on, but you may turn it off by using **-disable-build-dird** to prevent the Director from being built.

-enable-build-stored This option causes the make process to build the Storage daemon. By default, this option is on, but you may turn it off by using **-disable-build-stored** to prevent the Storage daemon from being built.

-enable-largefile This option (default) causes Bacula to be built with 64 bit file address support if it is available on your system. This permits Bacula to read and write files greater than 2 GBytes in size. You may disable this feature and revert to 32 bit file addresses by using **--disable-largefile**.

-disable-nls By default, Bacula uses the GNU Native Language Support (NLS) libraries. On some machines, these libraries may not be present or may not function correctly (especially on non-Linux implementations). In such cases, you may specify **-disable-nls** to disable use of those libraries. In such a case, Bacula will revert to using English.

-disable-ipv6 By default, Bacula enables IPv6 protocol. On some systems, the files for IPv6 may exist, but the functionality could be turned off in the kernel. In that case, in order to correctly build Bacula, you will explicitly need to use this option so that Bacula does not attempt to reference OS function calls that do not exist.

-with-mysql=<mysql-path> This enables building of the Catalog services for Bacula. It assumes that MySQL is running on your system, and expects it to be installed in the **mysql-path** that you specify. Normally, if MySQL is installed in a standard system location, you can simply use **-with-mysql** with no path specification. If you do use this option, please proceed to installing MySQL in the Installing and Configuring MySQL chapter before proceeding with the configuration.

See the note below under the -with-postgresql item.

-with-postgresql=<path> This provides an explicit path to the PostgreSQL libraries if Bacula cannot find it by default. Normally to build with PostgreSQL, you would simply use **-with-postgresql**.

Note, for Bacula to be configured properly, you must specify one of the two database options supported. That is: -with-postgresql, otherwise the ./configure will fail.

-with-openssl=<path> This configuration option is necessary if you want to enable TLS (ssl), which encrypts the communications within Bacula or if you want to use File Daemon PKI data encryption. Normally, the **path** specification is not necessary since the configuration searches for the OpenSSL libraries in standard system locations. However, you must ensure that all the libraries are loaded including **libssl-dev** or the equivalent on your system. Enabling OpenSSL in Bacula permits secure communications between the daemons and/or data encryption in the File daemon. For more information on using TLS, please see the Bacula TLS – Communications Encryption chapter of this manual. For more information on using PKI data encryption, please see the Bacula PKI – Data Encryption chapter of this manual.

If you get errors linking, you need to load the development libraries, or you need to disable SSL by setting without-openssl.

-with-libintl-prefix=<DIR> This option may be used to tell Bacula to search DIR/include and DIR/lib for the libintl headers and libraries needed for Native Language Support (NLS).

-enable-conio Tells Bacula to enable building the small, light weight readline replacement routine. It is generally much easier to configure than readline, although, like readline, it needs either the termcap or ncurses library.

-with-readline=<readline-path> Tells Bacula where **readline** is installed. Normally, Bacula will find readline if it is in a standard library. If it is not found and no -with-readline is specified, readline will be disabled. This option affects the Bacula build. Readline provides the Console program with a command line history and editing capability and is no longer supported, so you are on your own if you have problems.

-enable-readline Tells Bacula to enable readline support. It is normally disabled due to the large number of configuration problems and the fact that the package seems to change in incompatible ways from version to version.

-with-tcp-wrappers=<path> This specifies that you want TCP wrappers (man hosts_access(5)) compiled in. The path is optional since Bacula will normally find the libraries in the standard locations. This option affects the Bacula build. In specifying your restrictions in the **/etc/hosts.allow** or **/etc/hosts.deny** files, do not use the **twist** option (hosts_options(5)) or the Bacula process will be terminated. Note, when setting up your **/etc/hosts.allow** or **/etc/hosts.deny**, you must identify the Bacula daemon in question with the name you give it in your conf file rather than the name of the executable.

For more information on configuring and testing TCP wrappers, please see the Configuring and Testing TCP Wrappers section in the Security Chapter.

On SuSE, the libwrappers libraries needed to link Bacula are contained in the tcpd-devel package. On Red Hat, the package is named tcp_wrappers.

-with-archivedir=<path> The directory used for disk-based backups. Default value is /tmp. This parameter sets the default values in the bacula-dir.conf and bacula-sd.conf configuration files. For example, it sets the Where directive for the default restore job and the Archive Device directive for the FileStorage device.

This option is designed primarily for use in regression testing. Most users can safely ignore this option.

-with-working-dir=<working-directory-path> This option is mandatory and specifies a directory into which Bacula may safely place files that will remain between Bacula executions. For example, if the internal database is used, Bacula will keep those files in this directory. This option is only used to modify the daemon configuration files. You may also accomplish the same thing by directly editing them later. The working directory is not automatically created by the install process, so you must ensure that it exists before using Bacula for the first time.

-with-baseport=<port=number> In order to run, Bacula needs three TCP/IP ports (one for the Bacula Console, one for the Storage daemon, and one for the File daemon). The **--with-baseport** option will automatically assign three ports beginning at the base port address specified. You may also change the port number in the resulting configuration files. However, you need to take care that the numbers correspond correctly in each of the three daemon configuration files. The default base port is 9101, which assigns ports 9101 through 9103. These ports (9101, 9102, and 9103) have been officially assigned to Bacula by IANA. This option is only used to modify the daemon configuration files. You may also accomplish the same thing by directly editing them later.

-with-dump-email=<email-address> This option specifies the email address where any core dumps should be set. This option is normally only used by developers.

-with-pid-dir=<PATH> This specifies where Bacula should place the process id file during execution. The default is: **/var/run**. This directory is not created by the install process, so you must ensure that it exists before using Bacula the first time.

-with-subsys-dir=<PATH> This specifies where Bacula should place the subsystem lock file during execution. The default is **/var/run/subsys**. Please make sure that you do not specify the same directory for this directory and for the **sbindir** directory. This directory is used only within the autostart scripts. The subsys directory is not created by the Bacula install, so you must be sure to create it before using Bacula.

-with-dir-password=<Password> This option allows you to specify the password used to access the Director (normally from the Console program). If it is not specified, configure will automatically create a random password.

-with-fd-password=<Password> This option allows you to specify the password used to access the File daemon (normally called from the Director). If it is not specified, configure will automatically create a random password.

-with-sd-password=<Password> This option allows you to specify the password used to access the Storage daemon (normally called from the Director). If it is not specified, configure will automatically create a random password.

-with-dir-user=<User> This option allows you to specify the Userid used to run the Director. The Director must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the UserId specified on this option. If you specify this option, you must create the User prior to running **make install**, because the working directory owner will be set to **User**.

-with-dir-group=<Group> This option allows you to specify the GroupId used to run the Director. The Director must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the GroupId specified on this option. If you specify this option, you must create the Group prior to running **make install**, because the working directory group will be set to **Group**.

-with-sd-user=<User> This option allows you to specify the Userid used to run the Storage daemon. The Storage daemon must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the UserId specified on this option. If you use this option, you will need to take care that the Storage daemon has access to all the devices (tape drives, ...) that it needs.

-with-sd-group=<Group> This option allows you to specify the GroupId used to run the Storage daemon. The Storage daemon must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the GroupId specified on this option.

-with-fd-user=<User> This option allows you to specify the Userid used to run the File daemon. The File daemon must be started as root, and in most cases, it needs to run as root, so this option is used only in very special cases, after doing preliminary initializations, it can "drop" to the UserId specified on this option.

-with-fd-group=<Group> This option allows you to specify the GroupId used to run the File daemon. The File daemon must be started as root, and in most cases, it must be run as root, however, after doing preliminary initializations, it can "drop" to the GroupId specified on this option.

-with-mon-dir-password=<Password> This option allows you to specify the password used to access the Directory from the monitor. If it is not specified, configure will automatically create a random password.

-with-mon-fd-password=<Password> This option allows you to specify the password used to access the File daemon from the Monitor. If it is not specified, configure will automatically create a random password.

-with-mon-sd-password=<Password> This option allows you to specify the password used to access the Storage daemon from the Monitor. If it is not specified, configure will automatically create a random password.

-with-db-name=<database-name> This option allows you to specify the database name to be used in the conf files. The default is bacula.

-with-db-user=<database-user> This option allows you to specify the database user name to be used in the conf files. The default is bacula.

Note, many other options are presented when you do a **./configure --help**, but they are not implemented.

18.12 Recommended Options for Most Systems

For most systems, we recommend starting with the following options:

```
./configure \
  --enable-smartalloc \
  --sbindir=/opt/bacula/bin \
  --sysconfdir=/opt/bacula/etc \
  --with-pid-dir=/opt/bacula/working \
  --with-subsys-dir=/opt/bacula/working \
  --with-working-dir=/opt/bacula/working
```

If you want to install Bacula in an installation directory rather than run it out of the build directory (as developers will do most of the time), you should also include the --sbindir and --sysconfdir options with appropriate paths. Neither are necessary if you do not use "make install" as is the case for most development work. The install process will create the sbindir and sysconfdir if they do not exist, but it will not automatically create the pid-dir, subsys-dir, or working-dir, so you must ensure that they exist before running Bacula for the first time.

18.13 Red Hat

or

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=/opt/bacula/bin \
  --sysconfdir=/opt/bacula/etc \
  --enable-smartalloc \
  --with-mysql \
  --with-working-dir=/opt/bacula/working
  --with-pid-dir=/opt/bacula/working \
  --with-subsys-dir=/opt/bacula/working
  --enable-readline
```

or finally, a completely traditional Red Hat Linux install, which we do not recommend, because it make it harder to backup Bacula for disaster recovery purposes:

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=/usr/sbin \
  --sysconfdir=/etc/bacula \
  --with-scriptdir=/etc/bacula \
  --enable-smartalloc \
  --enable-bat \
  --with-mysql \
  --with-working-dir=/var/bacula \
  --with-pid-dir=/var/run \
  --enable-readline
```

Note, Bacula assumes that /var/bacula, /var/run, and /var/lock/subsys exist so it will not automatically create them during the install process.

18.14 Solaris

To build Bacula from source, you will need the following installed on your system (they are not by default): libiconv, gcc 3.3.2, stdc++, libgcc (for stdc++ and gcc_s libraries), make 3.8 or later.

You will probably also need to: Add /usr/local/bin to PATH and Add /usr/ccs/bin to PATH for ar.

It is possible to build Bacula on Solaris with the Solaris compiler, but we recommend using GNU C++ if possible.

A typical configuration command might look like:

```
#!/bin/sh
CFLAGS="-g" ./configure \
  --sbindir=/opt/bacula/bin \
  --sysconfdir=/opt/bacula/etc \
  --with-mysql \
  --enable-smartalloc \
  --with-pid-dir=/opt/bacula/working \
  --with-subsys-dir=/opt/bacula/working \
  --with-working-dir=/opt/bacula/working
```

Note, you may need to install the following packages to build Bacula from source:

```
SUNWbinutils,
SUNWarc,
SUNWhea,
SUNWGcc,
SUNWGnutls
SUNWGnutls-devel
SUNWGmake
SUNWgccruntime
```

```
SUNWlibgcrypt
SUNWzlib
SUNWzlibs
SUNWreadline
SUNWbinutilsS
SUNWGmakeS
SUNWlibm

export
PATH=/usr/bin::/usr/ccs/bin:/etc:/usr/openwin/bin:/usr/local/bin:/usr/sfw/bin:/opt/sfw/bin:/usr/ucb:/usr/sbin
```

If you have installed special software not normally in the Solaris libraries, such as OpenSSL, or the packages shown above, then you may need to add **/usr/sfw/lib** to the library search path. Probably the simplest way to do so is to run:

```
setenv LDFLAGS "-L/usr/sfw/lib -R/usr/sfw/lib"
```

Prior to running the ./configure command.

Alternatively, you can set the LD_LIBARY_PATH and/or the LD_RUN_PATH environment variables appropriately.

It is also possible to use the **crle** program to set the library search path. However, this should be used with caution.

18.15 FreeBSD

Please see: The FreeBSD Diary for a detailed description on how to make Bacula work on your system. In addition, users of FreeBSD prior to 4.9-STABLE dated Mon Dec 29 15:18:01 2003 UTC who plan to use tape devices, please see the Tape Testing chapter of the Problems manual for **important** information on how to configure your tape drive for compatibility with Bacula.

If you are using Bacula with MySQL, you should take care to compile MySQL with FreeBSD native threads rather than LinuxThreads, since Bacula is normally built with FreeBSD native threads rather than LinuxTreads. Mixing the two will probably not work.

18.16 Win32

To install the binary Win32 version of the File daemon please see the Win32 Installation Chapter in this document.

18.17 One File Configure Script

The following script could be used if you want to put everything in a single directory (except for the working directory):

```
#!/bin/sh
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --mandir=$HOME/bacula/bin \
    --enable-smartalloc \
    --enable-bat \
    --with-pid-dir=$HOME/bacula/bin/working \
```

```
    --with-subsys-dir=$HOME/bacula/bin/working \
    --with-mysql \
    --with-working-dir=$HOME/bacula/bin/working \
    --with-dump-email=$USER@your-site.com \
    --with-job-email=$USER@your-site.com \
    --with-smtp-host=mail.your-site.com
exit 0
```

You may also want to put the following entries in your **/etc/services** file as it will make viewing the connections made by Bacula easier to recognize (i.e. netstat -a):

```
bacula-dir      9101/tcp
bacula-fd       9102/tcp
bacula-sd       9103/tcp
```

18.18 Installing Bacula

Before setting up your configuration files, you will want to install Bacula in its final location. Simply enter:

```
make install
```

If you have previously installed Bacula, the old binaries will be overwritten, but the old configuration files will remain unchanged, and the "new" configuration files will be appended with a **.new**. Generally if you have previously installed and run Bacula you will want to discard or ignore the configuration files with the appended **.new**.

18.19 Building a File Daemon or Client

If you run the Director and the Storage daemon on one machine and you wish to back up another machine, you must have a copy of the File daemon for that machine. If the machine and the Operating System are identical, you can simply copy the Bacula File daemon binary file **bacula-fd** as well as its configuration file **bacula-fd.conf** then modify the name and password in the conf file to be unique. Be sure to make corresponding additions to the Director's configuration file (**bacula-dir.conf**).

If the architecture or the OS level are different, you will need to build a File daemon on the Client machine. To do so, you can use the same **./configure** command as you did for your main program, starting either from a fresh copy of the source tree, or using **make distclean** before the **./configure**.

Since the File daemon does not access the Catalog database, you can remove the **--with-mysql** ption, then add **--enable-client-only**. This will compile only the necessary libraries and the client programs and thus avoids the necessity of installing one or another of those database programs to build the File daemon. With the above option, you simply enter **make** and just the client will be built.

18.20 Auto Starting the Daemons

If you wish the daemons to be automatically started and stopped when your system is booted (a good idea), one more step is necessary. First, the ./configure process must recognize your system – that is it must be a supported platform and not **unknown**, then you must install the platform dependent files by doing:

```
(become root)
make install-autostart
```

Please note, that the auto-start feature is implemented only on systems that we officially support (currently, FreeBSD, Red Hat/Fedora Linux, and Solaris), and has only been fully tested on Fedora Linux.

The **make install-autostart** will cause the appropriate startup scripts to be installed with the necessary symbolic links. On Red Hat/Fedora Linux systems, these scripts reside in **/etc/rc.d/init.d/bacula-dir /etc/rc.d/init.d/bacula-fd**, and **/etc/rc.d/init.d/bacula-sd**. However the exact location depends on what operating system you are using.

If you only wish to install the File daemon, you may do so with:

```
make install-autostart-fd
```

18.21 Other Make Notes

To simply build a new executable in any directory, enter:

```
make
```

To clean out all the objects and binaries (including the files named 1, 2, or 3, which are development temporary files), enter:

```
make clean
```

To really clean out everything for distribution, enter:

```
make distclean
```

note, this cleans out the Makefiles and is normally done from the top level directory to prepare for distribution of the source. To recover from this state, you must redo the **./configure** in the top level directory, since all the Makefiles will be deleted.

To add a new file in a subdirectory, edit the Makefile.in in that directory, then simply do a **make**. In most cases, the make will rebuild the Makefile from the new Makefile.in. In some case, you may need to issue the **make** a second time. In extreme cases, cd to the top level directory and enter: **make Makefiles**.

To add dependencies:

```
make depend
```

The **make depend** appends the header file dependencies for each of the object files to Makefile and Makefile.in. This command should be done in each directory where you change the dependencies. Normally, it only needs to be run when you add or delete source or header files. **make depend** is normally automatically invoked during the configuration process.

To install:

```
make install
```

This not normally done if you are developing Bacula, but is used if you are going to run it to backup your system.

After doing a **make install** the following files will be installed on your system (more or less). The exact files and location (directory) for each file depends on your **./configure** command.

NOTE: it is quite probable that this list is out of date. But it is a starting point.

```
bacula
bacula-dir
bacula-dir.conf
bacula-fd
bacula-fd.conf
bacula-sd
bacula-sd.conf
bextract
bls
bscan
btape
btraceback
btraceback.gdb
bconsole
bconsole.conf
create_mysql_database
dbcheck
delete_catalog_backup
drop_bacula_tables
drop_mysql_tables
make_bacula_tables
make_catalog_backup
make_mysql_tables
mtx-changer
query.sql
bsmtp
startmysql
stopmysql
bwx-console
bwx-console.conf
9 man pages
```

18.22 Modifying the Bacula Configuration Files

See the chapter Configuring Bacula in this manual for instructions on how to set Bacula configuration files.

Chapter 19

Critical Items to Implement Before Production

We recommend you take your time before implementing a production a Bacula backup system since Bacula is a rather complex program, and if you make a mistake, you may suddenly find that you cannot restore your files in case of a disaster. This is especially true if you have not previously used a major backup product.

If you follow the instructions in this chapter, you will have covered most of the major problems that can occur. It goes without saying that if you ever find that we have left out an important point, please inform us, so that we can document it to the benefit of everyone.

19.1 Critical Items

The following assumes that you have installed Bacula, you more or less understand it, you have at least worked through the tutorial or have equivalent experience, and that you have set up a basic production configuration. If you haven't done the above, please do so and then come back here. The following is a sort of checklist that points with perhaps a brief explanation of why you should do it. In most cases, you will find the details elsewhere in the manual. The order is more or less the order you would use in setting up a production system (if you already are in production, use the checklist anyway).

- Test your tape drive for compatibility with Bacula by using the `test` command in the See the **btape** section (section 1.9 page 12) of the Bacula Community Edition Utility programs.
- Better than doing the above is to walk through the nine steps in the **Tape Testing** chapter (chapter 3 page 25) of the Bacula Community Edition Problems Resolution Guide. It may take you a bit of time, but it will eliminate surprises.
- Test the end of tape handling of your tape drive by using the `fill` command in the **btape program** section (section 1.9 page 12) (Part of the Bacula Community Edition Utility programs)
- If you are using a Linux 2.4 kernel, make sure that `/lib/tls` is disabled. Bacula does not work with this library. See the second point under Supported Operating Systems.
- Do at least one restore of files. If you backup multiple OS types (Linux, Solaris, HP, MacOS, FreeBSD, Win32, ...), restore files from each system type. The Restoring Files chapter shows you how.
- Write a bootstrap file to a separate system for each backup job. The **Write Bootstrap** directive is described in the Director Configuration chapter of the manual, and more details are available in the Bootstrap File chapter. Also, the default `bacula-dir.conf` comes

with a **Write Bootstrap** directive defined. This allows you to recover the state of your system as of the last backup.

- Backup your catalog. An example of this is found in the default `bacula-dir.conf` file. The backup script is installed by default and should handle any database, though you may want to make your own local modifications. See also Backing Up Your Bacula Database — Security Considerations for more information.
- Write a bootstrap file for the catalog. An example of this is found in the default `bacula-dir.conf` file. This will allow you to quickly restore your catalog in the event it is wiped out – otherwise it is many excruciating hours of work.
- Make a copy of the `bacula-dir.conf`, `bacula-sd.conf`, and `bacula-fd.conf` files that you are using on your server. Put it in a safe place (on another machine) as these files can be difficult to reconstruct if your server dies.
- Make a Bacula Rescue CDROM! See the Disaster Recovery Using a Bacula Rescue CDROM chapter. It is trivial to make such a CDROM, and it can make system recovery in the event of a lost hard disk infinitely easier.
- Bacula assumes all filenames are in UTF-8 format. This is important when saving the filenames to the catalog. For Win32 machine, Bacula will automatically convert from Unicode to UTF-8, but on Unix, Linux, *BSD, and MacOS X machines, you must explicitly ensure that your locale is set properly. Typically this means that the **LANG** environment variable must end in **.UTF-8**. A full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary.

  On most modern Win32 machines, you can edit the configuration files with **notepad** and choose output encoding UTF-8.

19.2 Recommended Items

Although these items may not be critical, they are recommended and will help you avoid problems.

- Read the Quick Start Guide to Bacula
- After installing and experimenting with Bacula, read and work carefully through the examples in the Tutorial chapter of this manual.
- Learn what each of the **Bacula Utility Programs** (chapter 1 page 1) does.
- Set up reasonable retention periods so that your catalog does not grow to be too big. See the following three chapters:
  - Recycling your Volumes,
  - Basic Volume Management,
  - Using Pools to Manage Volumes.
- Perform a bare metal recovery using the Bacula Rescue CDROM. See the Disaster Recovery Using a Bacula Rescue CDROM chapter.

If you absolutely must implement a system where you write a different tape each night and take it offsite in the morning. We recommend that you do several things:

- Write a bootstrap file of your backed up data and a bootstrap file of your catalog backup to a usb disk or a CDROM and take that with the tape. If this is not possible, try to write those files to another computer or offsite computer, or send them as email to a friend. If none of that is possible, at least print the bootstrap files and take that offsite with the tape. Having the bootstrap files will make recovery much easier.
- It is better not to force Bacula to load a particular tape each day. Instead, let Bacula choose the tape. If you need to know what tape to mount, you can print a list of recycled

and appendable tapes daily, and select any tape from that list. Bacula may propose a particular tape for use that it considers optimal, but it will accept any valid tape from the correct pool.

Chapter 20

A Brief Tutorial

This chapter will guide you through running Bacula. To do so, we assume you have installed Bacula, possibly in a single file as shown in the previous chapter, in which case, you can run Bacula as non-root for these tests. However, we assume that you have not changed the .conf files. If you have modified the .conf files, please go back and uninstall Bacula, then reinstall it, but do not make any changes. The examples in this chapter use the default configuration files, and will write the volumes to disk in your `/tmp` directory, in addition, the data backed up will be the source directory where you built Bacula. As a consequence, you can run all the Bacula daemons for these tests as non-root. Please note, in production, your File daemon(s) must run as root. See the Security chapter for more information on this subject.

The general flow of running Bacula is:

1. cd <install-directory>
2. Start the Database (if using `MySQL` or `PostgreSQL`)
3. Start the Daemons with `./bacula start`
4. Start the Console program to interact with the Director
5. Run a job
6. When the Volume fills, unmount the Volume, if it is a tape, label a new one, and continue running. In this chapter, we will write only to disk files so you won't need to worry about tapes for the moment.
7. Test recovering some files from the Volume just written to ensure the backup is good and that you know how to recover. Better test before disaster strikes
8. Add a second client.

Each of these steps is described in more detail below.

20.1 Before Running Bacula

Before running Bacula for the first time in production, we recommend that you run the `test` command in the `btape` program as described in the **Utility Program** chapter (chapter 1.9 page 12) of the Bacula Community Edition Utility programs. This will help ensure that Bacula functions correctly with your tape drive. If you have a modern HP, Sony, or Quantum DDS or DLT tape drive running on Linux or Solaris, you can probably skip this test as Bacula is well

tested with these drives and systems. For all other cases, you are **strongly** encouraged to run the test before continuing. btape also has a fill command that attempts to duplicate what Bacula does when filling a tape and writing on the next tape. You should consider trying this command as well, but be forewarned, it can take hours (about four hours on my drive) to fill a large capacity tape.

20.2 Starting the Database

If you are using MySQL or PostgreSQL as the Bacula database, you should start it before you attempt to run a job to avoid getting error messages from Bacula when it starts. Normally if you have installed the MySQL or PostgreSQL packages, the database server will automatically restart when your system is booted. In any case you must start the database server prior to starting the Bacula Director.

20.3 Starting the Daemons

Assuming you have built from source or have installed the rpms, to start the three daemons, from your installation directory, simply enter:

./bacula start

The bacula script starts the Storage daemon, the File daemon, and the Director daemon, which all normally run as daemons in the background. If you are using the autostart feature of Bacula, your daemons will either be automatically started on reboot, or you can control them individually with the files bacula-dir, bacula-fd, and bacula-sd, which are usually located in /etc/init.d, though the actual location is system dependent. Some distributions may do this differently.

Note, on Windows, currently only the File daemon is ported, and it must be started differently. Please see the Windows Version of Bacula Chapter of this manual.

The rpm packages configure the daemons to run as user=root and group=bacula. The rpm installation also creates the group bacula if it does not exist on the system. Any users that you add to the group bacula will have access to files created by the daemons. To disable or alter this behavior edit the daemon startup scripts:

- /etc/bacula/bacula
- /etc/init.d/bacula-dir
- /etc/init.d/bacula-sd
- /etc/init.d/bacula-fd

and then restart as noted above.

The installation chapter of this manual explains how you can install scripts that will automatically restart the daemons when the system starts.

20.4 Using the Director to Query and Start Jobs

To communicate with the director and to query the state of Bacula or run jobs, from the top level directory, simply enter:

./bconsole

Alternatively to running the command line console, if you have Qt4 installed and used the --**enable-bat** on the configure command, you may use the BAT;

```
./bat
```

Which has a graphical interface, and many more features than bconsole.

Two other possibilities are to run the GNOME console `bgnome-console` or the wxWidgets program `bwx-console`.

For simplicity, here we will describe only the `./bconsole` program. Most of what is described here applies equally well to `./bat`, `./bgnome-console`, and to `bwx-console`.

The `./bconsole` runs the Bacula Console program, which connects to the Director daemon. Since Bacula is a network program, you can run the Console program anywhere on your network. Most frequently, however, one runs it on the same machine as the Director. Normally, the Console program will print something similar to the following:

```
[kern@polymatou bin]$ ./bconsole
Connecting to Director lpmatou:9101
1000 OK: HeadMan Version: 2.1.8 (14 May 2007)
*
```

the asterisk is the console command prompt.

Type `help` to see a list of available commands:

```
*help
  Command       Description
  =======       ===========
  add           Add media to a pool
  autodisplay   Autodisplay console messages
  automount     Automount after label
  cancel        Cancel a job
  cloud         Specific Cloud commands
  create        Create DB Pool from resource
  delete        Delete volume, pool, client or job
  disable       Disable a job, attributes batch process
  enable        Enable a job, attributes batch process
  estimate      Performs FileSet estimate, listing gives full listing
  exit          Terminate Bconsole session
  gui           Non-interactive gui mode
  help          Print help on specific command
  label         Label a tape
  list          List objects from catalog
  llist         Full or long list like list command
  messages      Display pending messages
  memory        Print current memory usage
  mount         Mount storage
  prune         Prune expired records from catalog
  purge         Purge records from catalog
  quit          Terminate Bconsole session
  query         Query catalog
  dedup         Manage Global Deduplication Engine
  restore       Restore files
  relabel       Relabel a tape
  release       Release storage
  reload        Reload conf file
  run           Run a job
  restart       Restart a job
  resume        Resume a job
  status        Report status
  stop          Stop a job
  setdebug      Sets debug level
  setbandwidth  Sets bandwidth
  snapshot      Handle snapshots
  setip         Sets new client address -- if authorized
```

```
  show          Show resource records
  sqlquery      Use SQL to query catalog
  statistics    Display daemon statistics data
  tag           Manage tags
  time          Print current time
  trace         Turn on/off trace to file
  truncate      Truncate one or more Volumes
  unmount       Unmount storage
  umount        Umount - for old-time Unix guys, see unmount
  update        Update volume, pool, job or stats
  use           Use catalog xxx
  var           Does variable expansion
  version       Print Director version
  wait          Wait until no jobs are running

When at a prompt, entering a period cancels the command.
*
```

Details of the console program's commands are explained in the **Console** chapter (chapter 1 page 1) of the Bacula Community Edition Console manual.

20.5 Running a Job

At this point, we assume you have done the following:

- Configured Bacula with `./configure --:your-options`
- Built Bacula using `make`
- Installed Bacula using `make install`
- Have created your database with, for example, `./create_mysql_database`
- Have created the Bacula database tables with, `./make_bacula_tables`
- Have possibly edited your `bacula-dir.conf` file to personalize it a bit. BE CAREFUL! if you change the Director's name or password, you will need to make similar modifications in the other .conf files. For the moment it is probably better to make no changes.
- You have started Bacula with `./bacula start`
- You have invoked the Console program with `./bconsole`

Furthermore, we assume for the moment you are using the default configuration files.

At this point, enter the following command:

```
show filesets
```

and you should get something similar to:

```
FileSet: name=Full Set
      O M
      N
      I /home/kern/bacula/regress/build
      N
      E /proc
      E /tmp
      E /.journal
      E /.fsck
      N
FileSet: name=Catalog
      O M
      N
      I /home/kern/bacula/regress/working/bacula.sql
      N
```

This is a pre-defined **FileSet** that will backup the Bacula source directory. The actual directory names printed should correspond to your system configuration. For testing purposes, we have chosen a directory of moderate size (about 40 Megabytes) and complexity without being too big. The FileSet **Catalog** is used for backing up Bacula's catalog and is not of interest to us for the moment. The **I** entries are the files or directories that will be included in the backup and the **E** are those that will be excluded, and the **O** entries are the options specified for the FileSet. You can change what is backed up by editing `bacula-dir.conf` and changing the **File =** line in the **FileSet** resource.

Now is the time to run your first backup job. We are going to backup your Bacula source directory to a File Volume in your `/tmp` directory just to show you how easy it is. Now enter:

```
status dir
```

and you should get the following output:

```
rufus-dir Version: 1.30 (28 April 2003)
Daemon started 28-Apr-2003 14:03, 0 Jobs run.
Console connected at 28-Apr-2003 14:03
No jobs are running.
Level          Type     Scheduled          Name
===========================================================================
Incremental    Backup   29-Apr-2003 01:05  Client1
Full           Backup   29-Apr-2003 01:10  BackupCatalog
====
```

where the times and the Director's name will be different according to your setup. This shows that an Incremental job is scheduled to run for the Job **Client1** at 1:05am and that at 1:10, a **BackupCatalog** is scheduled to run. Note, you should probably change the name **Client1** to be the name of your machine, if not, when you add additional clients, it will be very confusing. For my real machine, I use **Rufus** rather than **Client1** as in this example.

Now enter:

```
status client
```

and you should get something like:

```
The defined Client resources are:
     1: rufus-fd
Item 1 selected automatically.
Connecting to Client rufus-fd at rufus:9102
rufus-fd Version: 1.30 (28 April 2003)
Daemon started 28-Apr-2003 14:03, 0 Jobs run.
Director connected at: 28-Apr-2003 14:14
No jobs running.
====
```

In this case, the client is named **rufus-fd** your name will be different, but the line beginning with **rufus-fd Version ...** is printed by your File daemon, so we are now sure it is up and running.

Finally do the same for your Storage daemon with:

```
status storage
```

and you should get:

```
The defined Storage resources are:
     1: File
Item 1 selected automatically.
Connecting to Storage daemon File at rufus:9103
rufus-sd Version: 1.30 (28 April 2003)
Daemon started 28-Apr-2003 14:03, 0 Jobs run.
Device /tmp is not open.
No jobs running.
====
```

You will notice that the default Storage daemon device is named **File** and that it will use device **/tmp**, which is not currently open.

Now, let's actually run a job with:

```
run
```

you should get the following output:

```
Using default Catalog name=MyCatalog DB=bacula
A job name must be specified.
The defined Job resources are:
     1: Client1
     2: BackupCatalog
     3: RestoreFiles
Select Job resource (1-3):
```

Here, Bacula has listed the three different Jobs that you can run, and you should choose number **1** and type enter, at which point you will get:

```
Run Backup job
JobName:  Client1
FileSet:  Full Set
Level:    Incremental
Client:   rufus-fd
Storage:  File
Pool:     Default
When:     2003-04-28 14:18:57
OK to run? (yes/mod/no):
```

At this point, take some time to look carefully at what is printed and understand it. It is asking you if it is OK to run a job named **Client1** with FileSet **Full Set** (we listed above) as an Incremental job on your Client (your client name will be different), and to use Storage **File** and Pool **Default**, and finally, it wants to run it now (the current time should be displayed by your console).

Here we have the choice to run (**yes**), to modify one or more of the above parameters (**mod**), or to not run the job (**no**). Please enter **yes**, at which point you should immediately get the command prompt (an asterisk). If you wait a few seconds, then enter the command `messages` you will get back something like:

```
28-Apr-2003 14:22 rufus-dir: Last FULL backup time not found. Doing
                  FULL backup.
28-Apr-2003 14:22 rufus-dir: Start Backup JobId 1,
                  Job=Client1.2003-04-28_14.22.33
28-Apr-2003 14:22 rufus-sd: Job Client1.2003-04-28_14.22.33 waiting.
                  Cannot find any appendable volumes.
Please use the "label"  command to create a new Volume for:
    Storage:      FileStorage
    Media type:   File
    Pool:         Default
```

The first message, indicates that no previous Full backup was done, so Bacula is upgrading our Incremental job to a Full backup (this is normal). The second message indicates that the job started with JobId 1., and the third message tells us that Bacula cannot find any Volumes in the Pool for writing the output. This is normal because we have not yet created (labeled) any Volumes. Bacula indicates to you all the details of the volume it needs.

At this point, the job is BLOCKED waiting for a Volume. You can check this if you want by doing a `status dir`. In order to continue, we must create a Volume that Bacula can write on. We do so with:

```
label
```

and Bacula will print:

```
The defined Storage resources are:
     1: File
Item 1 selected automatically.
Enter new Volume name:
```

at which point, you should enter some name beginning with a letter and containing only letters and numbers (period, hyphen, and underscore) are also permitted. For example, enter **TestVolume001**, and you should get back:

```
Defined Pools:
     1: Default
Item 1 selected automatically.
Connecting to Storage daemon File at rufus:9103 ...
Sending label command for Volume "TestVolume001" Slot 0 ...
3000 OK label. Volume=TestVolume001 Device=/tmp
Catalog record for Volume "TestVolume002", Slot 0  successfully created.
Requesting mount FileStorage ...
3001 OK mount. Device=/tmp
```

Finally, enter `messages` and you should get something like:

```
28-Apr-2003 14:30 rufus-sd: Wrote label to prelabeled Volume
   "TestVolume001" on device /tmp
28-Apr-2003 14:30 rufus-dir: \bacula{} 1.30 (28Apr03): 28-Apr-2003 14:30
JobId:                  1
Job:                    Client1.2003-04-28_14.22.33
FileSet:                Full Set
Backup Level:           Full
Client:                 rufus-fd
Start time:             28-Apr-2003 14:22
End time:               28-Apr-2003 14:30
Files Written:          1,444
Bytes Written:          38,988,877
Rate:                   81.2 KB/s
Software Compression:   None
Volume names(s):        TestVolume001
Volume Session Id:      1
Volume Session Time:    1051531381
Last Volume Bytes:      39,072,359
FD termination status:  OK
SD termination status:  OK
Termination:            Backup OK
28-Apr-2003 14:30 rufus-dir: Begin pruning Jobs.
28-Apr-2003 14:30 rufus-dir: No Jobs found to prune.
28-Apr-2003 14:30 rufus-dir: Begin pruning Files.
28-Apr-2003 14:30 rufus-dir: No Files found to prune.
28-Apr-2003 14:30 rufus-dir: End auto prune.
```

If you don't see the output immediately, you can keep entering `messages` until the job terminates, or you can enter, `autodisplay on` and your messages will automatically be displayed as soon as they are ready.

If you do an `ls -l` of your `/tmp` directory, you will see that you have the following item:

```
-rw-r-----    1 kern     kern     39072153 Apr 28 14:30 TestVolume001
```

This is the file Volume that you just wrote and it contains all the data of the job just run. If you run additional jobs, they will be appended to this Volume unless you specify otherwise.

You might ask yourself if you have to label all the Volumes that Bacula is going to use. The answer for disk Volumes, like the one we used, is no. It is possible to have Bacula automatically label volumes. For tape Volumes, you will most likely have to label each of the Volumes you want to use.

If you would like to stop here, you can simply enter `quit` in the Console program, and you can stop Bacula with `./bacula stop`. To clean up, simply delete the file `/tmp/TestVolume001`, and you should also re-initialize your database using:

```
./drop_bacula_tables
./make_bacula_tables
```

Please note that this will erase all information about the previous jobs that have run, and that you might want to do it now while testing but that normally you will not want to re-initialize your database.

If you would like to try restoring the files that you just backed up, read the following section.

20.6 Restoring Your Files

If you have run the default configuration and the save of the Bacula source code as demonstrated above, you can restore the backed up files in the Console program by entering:

```
restore all
```

where you will get:

```
First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified Job date
    13: Select object to restore
    14: Cancel
Select item:  (1-14):
```

As you can see, there are a number of options, but for the current demonstration, please enter **5** to do a restore of the last backup you did, and you will get the following output:

```
Defined Clients:
     1: rufus-fd
Item 1 selected automatically.
The defined FileSet resources are:
     1: 1  Full Set  2003-04-28 14:22:33
Item 1 selected automatically.
+-------+-------+----------+---------------------+---------------+
| JobId | Level | JobFiles | StartTime           | VolumeName    |
+-------+-------+----------+---------------------+---------------+
| 1     | F     | 1444     | 2003-04-28 14:22:33 | TestVolume002 |
+-------+-------+----------+---------------------+---------------+
You have selected the following JobId: 1
Building directory tree for JobId 1 ...
1 Job inserted into the tree and marked for extraction.
The defined Storage resources are:
     1: File
Item 1 selected automatically.
You are now entering file selection mode where you add and
remove files to be restored. All files are initially added.
Enter "done" to leave this mode.
cwd is: /
$
```

where I have truncated the listing on the right side to make it more readable. As you can see by starting at the top of the listing, Bacula knows what client you have, and since there was only one, it selected it automatically, likewise for the FileSet. Then Bacula produced a listing containing all the jobs that form the current backup, in this case, there is only one, and the Storage daemon was also automatically chosen. Bacula then took all the files that were in Job number 1 and entered them into a **directory tree** (a sort of in memory representation of your filesystem). At this point, you can use the `cd` and `ls` or `dir` commands to walk up and down the directory tree and view what files will be restored. For example, if I enter `cd /home/kern/bacula/bacula-1.30` and then enter `dir` I will get a listing of all the files in the Bacula source directory. On your system, the path will be somewhat different. For more information on this, please refer to the Restore Command Chapter of this manual for more details.

To exit this mode, simply enter:

```
done
```

and you will get the following output:

```
Bootstrap records written to
   /home/kern/bacula/testbin/working/restore.bsr
The restore job will require the following Volumes:

   TestVolume001
1444 files selected to restore.
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /home/kern/bacula/testbin/working/restore.bsr
Where:           /tmp/bacula-restores
Replace:         always
FileSet:         Full Set
Backup Client:   rufus-fd
Restore Client: rufus-fd
Storage:         File
JobId:           *None*
When:            2005-04-28 14:53:54
OK to run? (yes/mod/no):
```

If you answer **yes** your files will be restored to `/tmp/bacula-restores`. If you want to restore the files to their original locations, you must use the **mod** option and explicitly set **Where:** to nothing (or to /). We recommend you go ahead and answer **yes** and after a brief moment, enter `messages`, at which point you should get a listing of all the files that were restored as well as a summary of the job that looks similar to this:

```
28-Apr-2005 14:56 rufus-dir: \bacula{} 2.1.8 (08May07): 08-May-2007 14:56:06
Build OS:               i686-pc-linux-gnu suse 10.2
JobId:                  2
Job:                    RestoreFiles.2007-05-08_14.56.06
Restore Client:         rufus-fd
Start time:             08-May-2007 14:56
End time:               08-May-2007 14:56
Files Restored:         1,444
Bytes Restored:         38,816,381
Rate:                   9704.1 KB/s
FD Errors:              0
FD termination status:  OK
SD termination status:  OK
Termination:            Restore OK
08-May-2007 14:56 rufus-dir: Begin pruning Jobs.
08-May-2007 14:56 rufus-dir: No Jobs found to prune.
08-May-2007 14:56 rufus-dir: Begin pruning Files.
08-May-2007 14:56 rufus-dir: No Files found to prune.
08-May-2007 14:56 rufus-dir: End auto prune.
```

After exiting the Console program, you can examine the files in /tmp/bacula-restores, which will contain a small directory tree with all the files. Be sure to clean up at the end with:

```
rm -rf /tmp/bacula-restores
```

20.7 Quitting the Console Program

Simply enter the command quit.

20.8 Adding a Second Client

If you have gotten the example shown above to work on your system, you may be ready to add a second Client (File daemon). That is you have a second machine that you would like backed up. The only part you need installed on the other machine is the binary bacula-fd (or bacula-fd.exe for Windows) and its configuration file bacula-fd.conf. You can start with the same bacula-fd.conf file that you are currently using and make one minor modification to it to create the conf file for your second client. Change the File daemon name from whatever was configured, **rufus-fd** in the example above, but your system will have a different name. The best is to change it to the name of your second machine. For example:

```
...
#
# "Global" File daemon configuration specifications
#
FileDaemon {                          # this is me
  Name = rufus-fd
  FDport = 9102                  # where we listen for the director
  WorkingDirectory = /home/kern/bacula/working
  Pid Directory = /var/run
}
...
```

would become:

```
...
#
# "Global" File daemon configuration specifications
#
```

```
FileDaemon {                          # this is me
  Name = matou-fd
  FDport = 9102                  # where we listen for the director
  WorkingDirectory = /home/kern/bacula/working
  Pid Directory = /var/run
}
...
```

where I show just a portion of the file and have changed **rufus-fd** to **matou-fd**. The names you use are your choice. For the moment, I recommend you change nothing else. Later, you will want to change the password.

Now you should install that change on your second machine. Then you need to make some additions to your Director's configuration file to define the new File daemon or Client. Starting from our original example which should be installed on your system, you should add the following lines (essentially copies of the existing data but with the names changed) to your Director's configuration file `bacula-dir.conf`.

```
#
# Define the main nightly save backup job
#   By default, this job will back up to disk in /tmp
Job {
  Name = "Matou"
  Type = Backup
  Client = matou-fd
  FileSet = "Full Set"
  Schedule = "WeeklyCycle"
  Storage = File
  Messages = Standard
  Pool = Default
  Write Bootstrap = "/home/kern/bacula/working/matou.bsr"
}
# Client (File Services) to backup
Client {
  Name = matou-fd
  Address = matou
  FDPort = 9102
  Catalog = MyCatalog
  Password = "xxxxx"                  # password for
  File Retention = 30d                # 30 days
  Job Retention = 180d                # six months
  AutoPrune = yes                     # Prune expired Jobs/Files
}
```

Then make sure that the Address parameter in the Storage resource is set to the fully qualified domain name and not to something like "localhost". The address specified is sent to the File daemon (client) and it must be a fully qualified domain name. If you pass something like "localhost" it will not resolve correctly and will result in a time out when the File daemon fails to connect to the Storage daemon.

That is all that is necessary. I copied the existing resource to create a second Job (Matou) to backup the second client (matou-fd). It has the name **Matou**, the Client is named **matou-fd**, and the bootstrap file name is changed, but everything else is the same. This means that Matou will be backed up on the same schedule using the same set of tapes. You may want to change that later, but for now, let's keep it simple.

The second change was to add a new Client resource that defines **matou-fd** and has the correct address **matou**, but in real life, you may need a fully qualified domain name or an IP address. I also kept the password the same (shown as xxxxx for the example).

At this point, if you stop Bacula and restart it, and start the Client on the other machine, everything will be ready, and the prompts that you saw above will now include the second machine.

To make this a real production installation, you will possibly want to use different Pool, or a

different schedule. It is up to you to customize. In any case, you should change the password in both the Director's file and the Client's file for additional security.

For some important tips on changing names and passwords, and a diagram of what names and passwords must match, please see **Authorization Errors** chapter (chapter 1 page 2) in the Bacula Community Edition Problems Resolution Guide.

20.9 When The Tape Fills

If you have scheduled your job, typically nightly, there will come a time when the tape fills up and **Bacula** cannot continue. In this case, Bacula will send you a message similar to the following:

```
rufus-sd: block.c:337 === Write error errno=28: ERR=No space left
          on device
```

This indicates that Bacula got a write error because the tape is full. Bacula will then search the Pool specified for your Job looking for an appendable volume. In the best of all cases, you will have properly set your Retention Periods and you will have all your tapes marked to be Recycled, and **Bacula** will automatically recycle the tapes in your pool requesting and overwriting old Volumes. For more information on recycling, please see the Recycling chapter of this manual. If you find that your Volumes were not properly recycled (usually because of a configuration error), please see the Manually Recycling Volumes section of the Recycling chapter.

If like me, you have a very large set of Volumes and you label them with the date the Volume was first writing, or you have not set up your Retention periods, Bacula will not find a tape in the pool, and it will send you a message similar to the following:

```
rufus-sd: Job kernsave.2002-09-19.10:50:48 waiting. Cannot find any
          appendable volumes.
Please use the "label"  command to create a new Volume for:
    Storage:      SDT-10000
    Media type:   DDS-4
    Pool:         Default
```

Until you create a new Volume, this message will be repeated an hour later, then two hours later, and so on doubling the interval each time up to a maximum interval of one day.

The obvious question at this point is: What do I do now?

The answer is simple: first, using the Console program, close the tape drive using the `unmount` command. If you only have a single drive, it will be automatically selected, otherwise, make sure you release the one specified on the message (in this case **STD-10000**).

Next, you remove the tape from the drive and insert a new blank tape. Note, on some older tape drives, you may need to write an end of file mark (`mt -f /dev/nst0 weof`) to prevent the drive from running away when Bacula attempts to read the label.

Finally, you use the `label` command in the Console to write a label to the new Volume. The `label` command will contact the Storage daemon to write the software label, if it is successful, it will add the new Volume to the Pool, then issue a `mount` command to the Storage daemon. See the previous sections of this chapter for more details on labeling tapes.

The result is that Bacula will continue the previous Job writing the backup to the new Volume.

If you have a Pool of volumes and Bacula is cycling through them, instead of the above message "Cannot find any appendable volumes.", Bacula may ask you to mount a specific volume. In that case, you should attempt to do just that. If you do not have the volume any more (for any of a number of reasons), you can simply mount another volume from the same Pool, providing it

is appendable, and Bacula will use it. You can use the `list volumes` command in the console program to determine which volumes are appendable and which are not.

If like me, you have your Volume retention periods set correctly, but you have no more free Volumes, you can relabel and reuse a Volume as follows:

- Do a `list volumes` in the Console and select the oldest Volume for relabeling.
- If you have setup your Retention periods correctly, the Volume should have VolStatus **Purged**.
- If the VolStatus is not set to Purged, you will need to purge the database of Jobs that are written on that Volume. Do so by using the command `purge jobs volume` in the Console. If you have multiple Pools, you will be prompted for the Pool then enter the VolumeName (or MediaId) when requested.
- Then simply use the `relabel` command to relabel the Volume.

To manually relabel the Volume use the following additional steps:

- To delete the Volume from the catalog use the `delete volume` command in the Console and select the VolumeName (or MediaId) to be deleted.
- Use the `unmount` command in the Console to unmount the old tape.
- Physically relabel the old Volume that you deleted so that it can be reused.
- Insert the old Volume in the tape drive.
- From a command line do: `mt -f /dev/st0 rewind` and `mt -f /dev/st0 weof`, where you need to use the proper tape drive name for your system in place of `/dev/st0`.
- Use the `label` command in the Console to write a new Bacula label on your tape.
- Use the `mount` command in the Console if it is not automatically done, so that Bacula starts using your newly labeled tape.

20.10 Other Useful Console Commands

status dir Print a status of all running jobs and jobs scheduled in the next 24 hours.

status The console program will prompt you to select a daemon type, then will request the daemon's status.

status jobid=nn Print a status of JobId nn if it is running. The Storage daemon is contacted and requested to print a current status of the job as well.

list pools List the pools defined in the Catalog (normally only Default is used).

list media Lists all the media defined in the Catalog.

list jobs Lists all jobs in the Catalog that have run.

list jobid=nn Lists JobId nn from the Catalog.

list jobtotals Lists totals for all jobs in the Catalog.

list files jobid=nn List the files that were saved for JobId nn.

list jobmedia List the media information for each Job run.

messages Prints any messages that have been directed to the console.

unmount storage=storage-name Unmounts the drive associated with the storage device with the name **storage-name** if the drive is not currently being used. This command is used if you wish Bacula to free the drive so that you can use it to label a tape.

mount storage=storage-name Causes the drive associated with the storage device to be mounted again. When Bacula reaches the end of a volume and requests you to mount a new volume, you must issue this command after you have placed the new volume in the drive. In effect, it is the signal needed by Bacula to know to start reading or writing the new volume.

quit Exit or quit the console program.

Most of the commands given above, with the exception of `list`, will prompt you for the necessary arguments if you simply enter the command name.

20.11 Debug Daemon Output

If you want debug output from the daemons as they are running, start the daemons from the install directory as follows:

```
./bacula start -d100
```

This can be particularly helpful if your daemons do not start correctly, because direct daemon output to the console is normally directed to the NULL device, but with the debug level greater than zero, the output will be sent to the starting terminal.

To stop the three daemons, enter the following from the install directory:

```
./bacula stop
```

The execution of `bacula stop` may complain about pids not found. This is OK, especially if one of the daemons has died, which is very rare.

To do a full system save, each File daemon must be running as root so that it will have permission to access all the files. None of the other daemons require root privileges. However, the Storage daemon must be able to open the tape drives. On many systems, only root can access the tape drives. Either run the Storage daemon as root, or change the permissions on the tape devices to permit non-root access. `MySQL` and `PostgreSQL` can be installed and run with any userid; root privilege is not necessary.

20.12 Patience When Starting Daemons or Mounting Blank Tapes

When you start the Bacula daemons, the Storage daemon attempts to open all defined storage devices and verify the currently mounted Volume (if configured). Until all the storage devices are verified, the Storage daemon will not accept connections from the Console program. If a tape was previously used, it will be rewound, and on some devices this can take several minutes. As a consequence, you may need to have a bit of patience when first contacting the Storage daemon after starting the daemons. If you can see your tape drive, once the lights stop flashing, the drive will be ready to be used.

The same considerations apply if you have just mounted a blank tape in a drive such as an HP DLT. It can take a minute or two before the drive properly recognizes that the tape is blank. If you attempt to `mount` the tape with the Console program during this recognition period, it is quite possible that you will hang your SCSI driver (at least on my Red Hat Linux system). As a consequence, you are again urged to have patience when inserting blank tapes. Let the device settle down before attempting to access it.

20.13 Difficulties Connecting from the FD to the SD

If you are having difficulties getting one or more of your File daemons to connect to the Storage daemon, it is most likely because you have not used a fully qualified domain name on the **Address** directive in the Director's Storage resource. That is the resolver on the File daemon's machine (not on the Director's) must be able to resolve the name you supply into an IP address. An example of an address that is guaranteed not to work: **localhost**. An example that may work: **megalon**. An example that is more likely to work: **magalon.mydomain.com**. On Win32 if you don't have a good resolver (often true on older Win98 systems), you might try using an IP address in place of a name.

If your address is correct, then make sure that no other program is using the port 9103 on the Storage daemon's machine. The Bacula port numbers are authorized by IANA, and should not be used by other programs, but apparently some HP printers do use these port numbers. A `netstat -a` on the Storage daemon's machine can determine who is using the 9103 port (used for FD to SD communications in Bacula).

20.14 Daemon Command Line Options

Each of the three daemons (Director, File, Storage) accepts a small set of options on the command line. In general, each of the daemons as well as the Console program accepts the following options:

-c <file> Define the file to use as a configuration file. The default is the daemon name followed by **.conf** i.e. `bacula-dir.conf` for the Director, `bacula-fd.conf` for the File daemon, and `bacula-sd.conf` for the Storage daemon.

-d nn Set the debug level to **nn**. Higher levels of debug cause more information to be displayed on STDOUT concerning what the daemon is doing.

-f Run the daemon in the foreground. This option is needed to run the daemon under the debugger.

-g <group> Run the daemon under this group. This must be a group name, not a GID.

-s Do not trap signals. This option is needed to run the daemon under the debugger.

-t Read the configuration file and print any error messages, then immediately exit. Useful for syntax testing of new configuration files.

-u <user> Run the daemon as this user. This must be a user name, not a UID.

-v Be more verbose or more complete in printing error and informational messages. Recommended.

-? Print the version and list of options.

20.15 Creating a Pool

Creating the Pool is automatically done when **Bacula** starts, so if you understand Pools, you can skip to the next section.

When you run a job, one of the things that Bacula must know is what Volumes to use to backup the FileSet. Instead of specifying a Volume (tape) directly, you specify which Pool of Volumes you want Bacula to consult when it wants a tape for writing backups. Bacula will select the first available Volume from the Pool that is appropriate for the Storage device you have specified for the Job being run. When a volume has filled up with data, **Bacula** will change its VolStatus

from **Append** to **Full**, and then **Bacula** will use the next volume and so on. If no appendable Volume exists in the Pool, the Director will attempt to recycle an old Volume, if there are still no appendable Volumes available, **Bacula** will send a message requesting the operator to create an appropriate Volume.

Bacula keeps track of the Pool name, the volumes contained in the Pool, and a number of attributes of each of those Volumes.

When Bacula starts, it ensures that all Pool resource definitions have been recorded in the catalog. You can verify this by entering:

```
list pools
```

to the console program, which should print something like the following:

```
*list pools
Using default Catalog name=MySQL DB=bacula
+--------+---------+---------+---------+----------+-------------+
| PoolId | Name    | NumVols | MaxVols | PoolType | LabelFormat |
+--------+---------+---------+---------+----------+-------------+
| 1      | Default | 3       | 0       | Backup   | *           |
| 2      | File    | 12      | 12      | Backup   | File        |
+--------+---------+---------+---------+----------+-------------+
*
```

If you attempt to create the same Pool name a second time, **Bacula** will print:

```
Error: Pool Default already exists.
Once created, you may use the \bcommandname{update} command to
modify many of the values in the Pool record.
```

20.16 Labeling Your Volumes

Bacula requires that each Volume contains a software label. There are several strategies for labeling volumes. The one I use is to label them as they are needed by **Bacula** using the console program. That is when Bacula needs a new Volume, and it does not find one in the catalog, it will send me an email message requesting that I add Volumes to the Pool. I then use the `label` command in the Console program to label a new Volume and to define it in the Pool database, after which Bacula will begin writing on the new Volume. Alternatively, I can use the Console `relabel` command to relabel a Volume that is no longer used providing it has VolStatus **Purged**.

Another strategy is to label a set of volumes at the start, then use them as **Bacula** requests them. This is most often done if you are cycling through a set of tapes, for example using an autochanger. For more details on recycling, please see the Automatic Volume Recycling chapter of this manual.

If you run a Bacula job, and you have no labeled tapes in the Pool, Bacula will inform you, and you can create them "on-the-fly" so to speak. In my case, I label my tapes with the date, for example: **DLT-18April02**. See below for the details of using the `label` command.

20.17 Labeling Volumes with the Console Program

Labeling volumes is normally done by using the console program.

1. ./bconsole
2. label

If Bacula complains that you cannot label the tape because it is already labeled, simply `unmount` the tape using the `unmount` command in the console, then physically mount a blank tape and re-issue the `label` command.

Since the physical storage media is different for each device, the `label` command will provide you with a list of the defined Storage resources such as the following:

```
The defined Storage resources are:
     1: File
     2: 8mmDrive
     3: DLTDrive
     4: SDT-10000
Select Storage resource (1-4):
```

At this point, you should have a blank tape in the drive corresponding to the Storage resource that you select.

It will then ask you for the Volume name.

```
Enter new Volume name:
```

If Bacula complains:

```
Media record for Volume xxxx already exists.
```

It means that the volume name **xxxx** that you entered already exists in the Media database. You can list all the defined Media (Volumes) with the `list media` command. Note, the LastWritten column has been truncated for proper printing.

```
+---------------+---------+--------+----------------+-----/~/-+------------+-----+
| VolumeName    | MediaTyp| VolStat| VolBytes       | LastWri | VolReten   | Recy|
+---------------+---------+--------+----------------+---------+------------+-----+
| DLTVol0002    | DLT8000 | Purged | 56,128,042,217 | 2001-10 | 31,536,000 |   0 |
| DLT-07Oct2001 | DLT8000 | Full   | 56,172,030,586 | 2001-11 | 31,536,000 |   0 |
| DLT-08Nov2001 | DLT8000 | Full   | 55,691,684,216 | 2001-12 | 31,536,000 |   0 |
| DLT-01Dec2001 | DLT8000 | Full   | 55,162,215,866 | 2001-12 | 31,536,000 |   0 |
| DLT-28Dec2001 | DLT8000 | Full   | 57,888,007,042 | 2002-01 | 31,536,000 |   0 |
| DLT-20Jan2002 | DLT8000 | Full   | 57,003,507,308 | 2002-02 | 31,536,000 |   0 |
| DLT-16Feb2002 | DLT8000 | Full   | 55,772,630,824 | 2002-03 | 31,536,000 |   0 |
| DLT-12Mar2002 | DLT8000 | Full   | 50,666,320,453 | 1970-01 | 31,536,000 |   0 |
| DLT-27Mar2002 | DLT8000 | Full   | 57,592,952,309 | 2002-04 | 31,536,000 |   0 |
| DLT-15Apr2002 | DLT8000 | Full   | 57,190,864,185 | 2002-05 | 31,536,000 |   0 |
| DLT-04May2002 | DLT8000 | Full   | 60,486,677,724 | 2002-05 | 31,536,000 |   0 |
| DLT-26May02   | DLT8000 | Append |  1,336,699,620 | 2002-05 | 31,536,000 |   1 |
+---------------+---------+--------+----------------+-----/~/-+------------+-----+
```

Once Bacula has verified that the volume does not already exist, it will prompt you for the name of the Pool in which the Volume (tape) is to be created. If there is only one Pool (Default), it will be automatically selected.

If the tape is successfully labeled, a Volume record will also be created in the Pool. That is the Volume name and all its other attributes will appear when you list the Pool. In addition, that Volume will be available for backup if the MediaType matches what is requested by the Storage daemon.

When you labeled the tape, you answered very few questions about it – principally the Volume name, and perhaps the Slot. However, a Volume record in the catalog database (internally

known as a Media record) contains quite a few attributes. Most of these attributes will be filled in from the default values that were defined in the Pool (i.e. the Pool holds most of the default attributes used when creating a Volume).

It is also possible to add media to the pool without physically labeling the Volumes. This can be done with the `add` command. For more information, please see the **Console** chapter (chapter 1 page 1) of the Bacula Community Edition Console manual.

Chapter 21

Customizing the Configuration Files

When each of the Bacula programs starts, it reads a configuration file specified on the command line or the default `bacula-dir.conf`, `bacula-fd.conf`, `bacula-sd.conf`, or `console.conf` for the Director daemon, the File daemon, the Storage daemon, and the Console program respectively.

Each service (Director, Client, Storage, Console) has its own configuration file containing a set of Resource definitions. These resources are very similar from one service to another, but may contain different directives (records) depending on the service. For example, in the Director's resource file, the **`Director`** resource defines the name of the Director, a number of global Director parameters and his password. In the File daemon configuration file, the **`Director`** resource specifies which Directors are permitted to use the File daemon.

Before running Bacula for the first time, you must customize the configuration files for each daemon. Default configuration files will have been created by the installation process, but you will need to modify them to correspond to your system. An overall view of the resources can be seen in the following:

21.1 Character Sets

Bacula is designed to handle most character sets of the world, US ASCII, German, French, Chinese, ... However, it does this by encoding everything in UTF-8, and it expects all configuration files (including those read on Win32 machines) to be in UTF-8 format. UTF-8 is typically the default on Linux machines, but not on all Unix machines, nor on Windows, so you must take some care to ensure that your locale is set properly before starting Bacula.

To ensure that Bacula configuration files can be correctly read including foreign characters the **LANG** environment variable must end in **.UTF-8**. An full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary. On most newer Win32 machines, you can use `notepad` to edit the conf files, then choose output encoding UTF-8.

Bacula assumes that all filenames are in UTF-8 format on Linux and Unix machines. On Win32 they are in Unicode (UTF-16), and will be automatically converted to UTF-8 format.

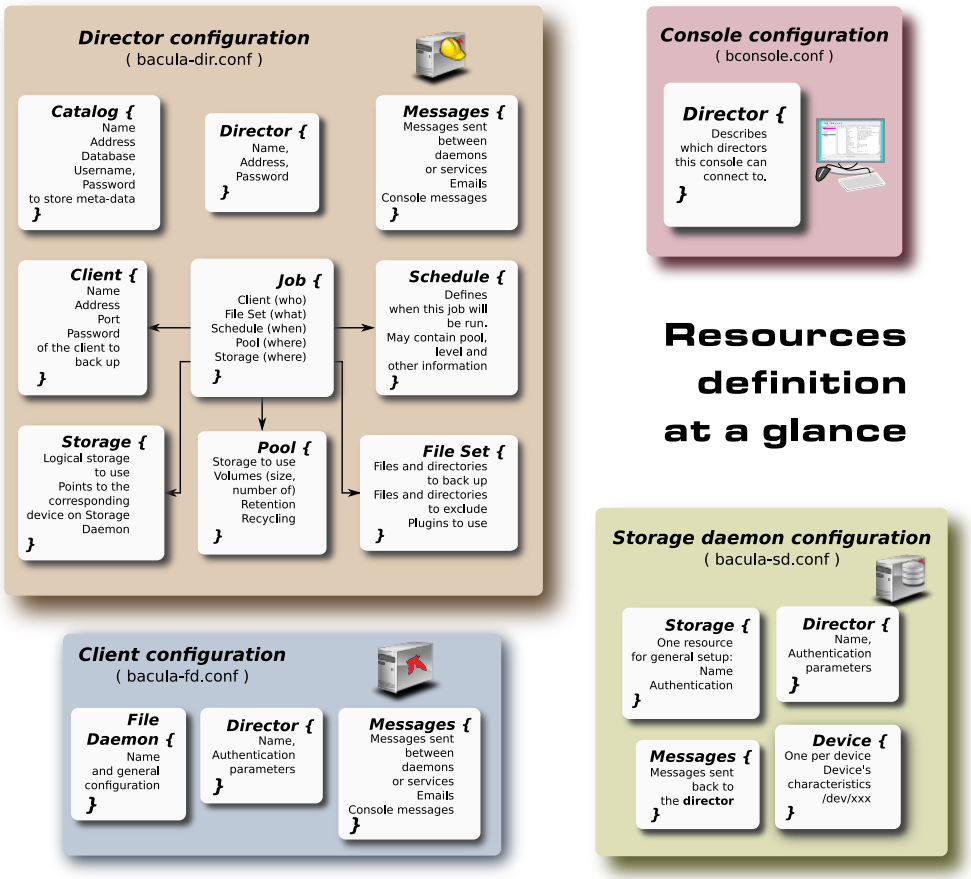


Figure 21.1: Bacula Objects

21.2 Resource Directive Format

Although, you won't need to know the details of all the directives a basic knowledge of Bacula resource directives is essential. Each directive contained within the resource (within the braces) is composed of a keyword followed by an equal sign (=) followed by one or more values. The keywords must be one of the known Bacula resource record keywords, and it may be composed of upper or lower case characters and spaces.

Each resource definition MUST contain a Name directive, and may optionally contain a Description directive. The Name directive is used to uniquely identify the resource. The Description directive is (will be) used during display of the Resource to provide easier human recognition. For example:

```
Director {
  Name = "MyDir"
  Description = "Main Bacula Director"
  WorkingDirectory = "$HOME/bacula/bin/working"
}
```

Defines the Director resource with the name "MyDir" and a working directory `$HOME/bacula/bin/working`. In general, if you want spaces in a name to the right of the first equal sign (=), you must enclose that name within double quotes. Otherwise quotes are not generally necessary because once defined, quoted strings and unquoted strings are all equal.

21.2.1 Comments

When reading the configuration file, blank lines are ignored and everything after a hash sign (#) until the end of the line is taken to be a comment. A semicolon (;) is a logical end of line, and anything after the semicolon is considered as the next statement. If a statement appears on a line by itself, a semicolon is not necessary to terminate it, so generally in the examples in this manual, you will not see many semicolons.

21.2.2 Upper and Lower Case and Spaces

Case (upper/lower) and spaces are totally ignored in the resource directive keywords (the part before the equal sign).

Within the keyword (i.e. before the equal sign), spaces are not significant. Thus the keywords: **name**, **Name**, and **N a m e** are all identical.

Spaces after the equal sign and before the first character of the value are ignored.

In general, spaces within a value are significant (not ignored), and if the value is a name, you must enclose the name in double quotes for the spaces to be accepted. Names may contain up to 127 characters. Currently, a name may contain any ASCII character. Within a quoted string, any character following a backslash (\) is taken as itself (handy for inserting backslashes and double quotes (")).

Please note, however, that Bacula resource names as well as certain other names (e.g. Volume names) must contain only letters (including ISO accented letters), numbers, and a few special characters (space, underscore, dash, dot, colon). All other characters and punctuation are invalid.

21.2.3 Including other Configuration Files

If you wish to break your configuration file into smaller pieces, you can do so by including other files using the syntax `@filename` where `filename` is the full path and filename of another file.

The @filename specification can be given anywhere a primitive token would appear.

If you wish include all files in a specific directory, you can use the following:

```
# Include subfiles associated with configuration of clients.
# They define the bulk of the Clients, Jobs, and FileSets.
# Remember to "reload" the Director after adding a client file.
#
@|"sh -c 'for f in /etc/bacula/clientdefs/*.conf ; do echo @${f} ; done'"
```

21.2.4 Recognized Primitive Data Types

When parsing the resource directives, Bacula classifies the data according to the types listed below. The first time you read this, it may appear a bit overwhelming, but in reality, it is all pretty logical and straightforward.

name A keyword or name consisting of alphanumeric characters, including the hyphen, underscore, and dollar characters. The first character of a **name** must be a letter. A name has a maximum length currently set to 127 bytes. Typically keywords appear on the left side of an equal (i.e. they are Bacula keywords – i.e. Resource names or directive names). Keywords may not be quoted.

name-string A name-string is similar to a name, except that the name may be quoted and can thus contain additional characters including spaces. Name strings are limited to 127 characters in length. Name strings are typically used on the right side of an equal (i.e. they are values to be associated with a keyword).

string A quoted string containing virtually any character including spaces, or a non-quoted string. A string may be of any length. Strings are typically values that correspond to filenames, directories, or system command names. A backslash (\) turns the next character into itself, so to include a double quote in a string, you precede the double quote with a backslash. Likewise to include a backslash.

directory A directory is either a quoted or non-quoted string. A directory will be passed to your standard shell for expansion when it is scanned. Thus constructs such as $HOME are interpreted to be their correct values.

password This is a Bacula password and it is stored internally in MD5 hashed format.

integer A 32 bit integer value. It may be positive or negative.

positive integer A 32 bit positive integer value.

long integer A 64 bit integer value. Typically these are values such as bytes that can exceed 4 billion and thus require a 64 bit value.

<yes|no> Either a **yes** or a **no**.

size A size specified as bytes. Typically, this is a floating point scientific input format followed by an optional modifier. The floating point input is stored as a 64 bit integer value. If a modifier is present, it must immediately follow the value with no intervening spaces. The following modifiers are permitted:

k 1,024 (kilobytes)

kb 1,000 (kilobytes)

m 1,048,576 (megabytes)

mb 1,000,000 (megabytes)

g 1,073,741,824 (gigabytes)

gb 1,000,000,000 (gigabytes)

time A time or duration specified in seconds. The time is stored internally as a 64 bit integer value, but it is specified in two parts: a number part and a modifier part. The number can be an integer or a floating point number. If it is entered in floating point notation, it will be rounded to the nearest integer. The modifier is mandatory and follows the number part, either with or without intervening spaces. The following modifiers are permitted:

seconds seconds

minutes minutes (60 seconds)

hours hours (3600 seconds)

days days (3600*24 seconds)

weeks weeks (3600*24*7 seconds)

months months (3600*24*30 seconds)

quarters quarters (3600*24*91 seconds)

years years (3600*24*365 seconds)

Any abbreviation of these modifiers is also permitted (i.e. **seconds** may be specified as **sec** or **s**). A specification of **m** will be taken as months.

The specification of a time may have as many number/modifier parts as you wish. For example:

```
1 week 2 days 3 hours 10 mins
1 month 2 days 30 sec
```

are valid date specifications.

21.3 Resource Types

The table 21.1 lists all current Bacula resource types. It shows what resources must be defined for each service (daemon). The default configuration files will already contain at least one example of each permitted resource, so you need not worry about creating all these kinds of resources from scratch.

Table 21.1: Resource types

Resource	Director	Client	Storage	Console
Autochanger	No	No	Yes	No
Catalog	Yes	No	No	No
Client	Yes	Yes	No	No
Console	Yes	No	No	Yes
Device	No	No	Yes	No
Director	Yes	Yes	Yes	Yes
FileSet	Yes	No	No	No
Job	Yes	No	No	No
JobDefs	Yes	No	No	No
Message	Yes	Yes	Yes	No
Pool	Yes	No	No	No
Schedule	Yes	No	No	No
Storage	Yes	No	Yes	No

21.4 Names, Passwords and Authorization

In order for one daemon to contact another daemon, it must authorize itself with a password. In most cases, the password corresponds to a particular name, so both the name and the password must match to be authorized. Passwords are plain text, any text. They are not generated by any special process; just use random text.

The default configuration files are automatically defined for correct authorization with random passwords. If you add to or modify these files, you will need to take care to keep them consistent.

Here is sort of a picture of what names/passwords in which files/Resources must match up:

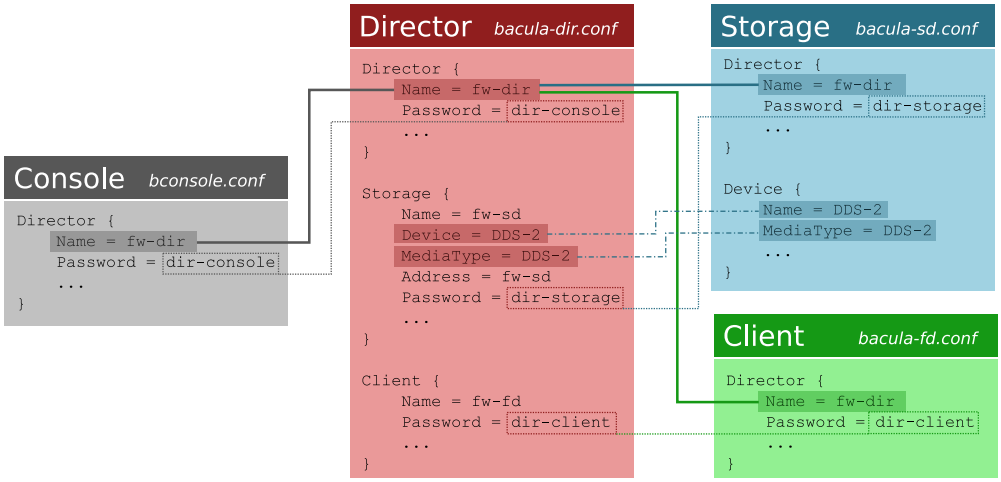


Figure 21.2: Configuration diagram

In the left column, you will find the Director, Storage, and Client resources, with their names and passwords – these are all in `bacula-dir.conf`. In the right column are where the corresponding values should be found in the Console, Storage daemon (SD), and File daemon (FD) configuration files.

Please note that the Address, **fd-sd**, that appears in the Storage resource of the Director, preceded with and asterisk in the above example, is passed to the File daemon in symbolic form. The File daemon then resolves it to an IP address. For this reason, you must use either an IP address or a fully qualified name. A name such as **localhost**, not being a fully qualified name, will resolve in the File daemon to the localhost of the File daemon, which is most likely not what is desired. The password used for the File daemon to authorize with the Storage daemon is a temporary password unique to each Job created by the daemons and is not specified in any .conf file.

21.5 Detailed Information for each Daemon

The details of each Resource and the directives permitted therein are described in the following chapters.

The following configuration files must be defined:

- Console – to define the resources for the Console program (user interface to the Director). It defines which Directors are available so that you may interact with them.
- Director – to define the resources necessary for the Director. You define all the Clients and Storage daemons that you use in this configuration file.

- Client – to define the resources for each client to be backed up. That is, you will have a separate Client resource file on each machine that runs a File daemon.
- Storage – to define the resources to be used by each Storage daemon. Normally, you will have a single Storage daemon that controls your tape drive or tape drives. However, if you have tape drives on several machines, you will have at least one Storage daemon per machine.

Chapter 22

Configuring the Director

Of all the configuration files needed to run Bacula, the Director's is the most complicated, and the one that you will need to modify the most often as you add clients or modify the FileSets.

For a general discussion of configuration files and resources including the data types recognized by Bacula, please see the Configuration chapter of this manual.

22.1 Director Resource Types

Director resource types may be the following:

Job, JobDefs, Client, Storage/Autochanger, Catalog, Schedule, FileSet, Pool, Director, Console, Counter, and Messages. We present them here in the most logical order for defining them.

Note, everything revolves around a job and is tied to a job in one way or another.

- Director – to define the Director's name and its access password used for authenticating the Console program. Only a single Director resource definition may appear in the Director's configuration file. If you have either `/dev/random` or `bc` on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.
- Job – to define the backup/restore Jobs and to tie together the Client, FileSet and Schedule resources to be used for each Job. Normally, you will have Jobs of different names corresponding to each client (i. e., one Job per client, but a different one with a different name for each client).
- JobDefs – optional resource for providing defaults for Job resources.
- Schedule – to define when a Job is to be automatically run by Bacula's internal scheduler. You may have any number of Schedules, but each job will reference only one.
- FileSet – to define the set of files to be backed up with each job. You may have any number of FileSets but each Job will reference only one.
- Client – to define what Client is to be backed up. You will generally have multiple Client definitions. Each Job will reference only a single client.
- Storage (or Autochanger) – to define on what physical device the Volumes should be mounted. You may have one or more Storage definitions.
- Pool – to define the pool of Volumes that can be used for a particular Job.If there is a large number of clients or volumes, you may want to have multiple Pools. Pools allow you to restrict a Job (or a Client) to use only a particular set of Volumes.

- Catalog – to define in what database to keep the list of files and the Volume names where they are backed up. Most people only use a single catalog. However, if you want to scale the Director to many clients, multiple catalogs can be helpful. Multiple catalogs require a bit more management because in general you must know what catalog contains what data. Currently, all Pools are defined in each catalog. This restriction will be removed in a later release.
- Console – to define how the administrator or user can interact with the Director.
- Counter – to define a counter variable that can be accessed by variable expansion used for creating Volume labels.
- Messages – to define where error and information messages are to be sent or logged. You may define multiple different message resources and hence direct particular classes of messages to different users or locations (files, ...).

22.2 The Director Resource

The Director resource defines the attributes of the Directors running on the network. In the current implementation, there is only a single Director resource, but a future design may contain multiple Directors to maintain index and media database redundancy.

Director Start of the Director resource. One and only one director resource must be supplied.

Name = <name> The director name used by the system administrator. This directive is required.

Description = <text> The text field contains a description of the Director that will be displayed in the graphical user interface. This directive is optional.

Password = <UA-password> Specifies the password that must be supplied for the default Bacula Console to be authorized. The same password must appear in the `Director` resource of the Console configuration file. For added security, the password is never passed across the network but instead a challenge response hash code created from the password. This directive is required. If you have either `/dev/random` or `bc` on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank and you must manually supply it.

The password is plain text. It is not generated through any special process but as noted above, it is better to use random text for security reasons.

Messages = <Messages-resource-name> The messages resource specifies where to deliver Director messages that are not associated with a specific Job. Most messages are specific to a job and will be directed to the Messages resource specified by the job. However, there are a few messages that can occur when no job is running. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its status files. This directory should be used only by Bacula but may be shared by other Bacula daemons. However, please note, if this directory is shared with other Bacula daemons (the File daemon and Storage daemon), you must ensure that the **Name** given to each daemon is unique so that the temporary filenames used do not collide. By default the Bacula configure process creates unique daemon names by postfixing them with `-dir`, `-fd`, and `-sd`. Standard shell expansion of the **Working Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded. This directive is required.

The working directory specified must already exist and be readable and writable by the Bacula daemon referencing it.

If you have specified a Director user and/or a Director group on your `./configure` line with `--with-dir-user` and/or `--with-dir-group` the Working Directory owner and group will be set to those values.

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its process Id file. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. Standard shell expansion of the **Pid Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

The PID directory specified must already exist and be readable and writable by the Bacula daemon referencing it

Typically on Linux systems, you will set this to: `/var/run`. If you are not installing Bacula in the system directories, you can use the `Working Directory` as defined above. This directive is required.

Scripts Directory = <Directory> This directive is optional and, if defined, specifies a directory in which the Director and the Storage daemon will look for many of the scripts that it needs to use during particular operations such as starting/stopping, the `mtx-changer` script, tape alerts, as well as catalog updates. This directory may be shared by other Bacula daemons. Standard shell expansion of the directory is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

QueryFile = <Path> This directive is mandatory and specifies a directory and file in which the Director can find the canned SQL statements for the `query` command of the Console. Standard shell expansion of the <Path> is done when the configuration file is read so that values such as **$HOME** will be properly expanded. This directive is required.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Client resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the `setsockopt` `TCP_KEEPIDLE` function. The default value is 300s.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of total Director Jobs that should run concurrently. The default is set to 20, but you may set it to a larger number. Every valid connection to any daemon (Director, File daemon, or Storage daemon) results in a Job. This includes connections from Bacula **Console**. Thus the number of concurrent Jobs must, in general, be greater than the maximum number of Jobs that you wish to actually run.

In general, increasing the number of Concurrent Jobs increases the total throughtput of Bacula, because the simultaneous Jobs can all feed data to the Storage daemon and to the Catalog at the same time. However, keep in mind, that the Volume format becomes more complicated with multiple simultaneous jobs, consequently, restores may take longer if Bacula must sort through interleaved volume blocks from multiple simultaneous jobs. Though not normally necessary, this can be avoided by having each simultaneous job write to a different volume or by using data spooling, which will first spool the data to disk simultaneously, then write one spool file at a time to the volume thus avoiding excessive interleaving of the different job blocks.

FD Connect Timeout = <time> where <time> is the time that the Director should continue attempting to contact the File daemon to start a job, and after which the Director will cancel the job. The default is 3 minutes.

SD Connect Timeout = <time> where <time> is the time that the Director should continue attempting to contact the Storage daemon to start a job, and after which the Director will cancel the job. The default is 30 minutes.

DirAddresses = <IP-address-specification> Specify the ports and addresses on which the Director daemon will listen for Bacula Console connections. Probably the simplest way to explain this is to show an example:

```
DirAddresses  = {
  ip = { addr = 1.2.3.4; port = 1205;}
  ipv4 = {
    addr = 1.2.3.4; port = http;
  }
```

```
    ipv6 = {
      addr = 1.2.3.4;
      port = 1205;
    }
    ip = {
      addr = 1.2.3.4
      port = 1205
    }
    ip = { addr = 1.2.3.4 }
    ip = { addr = 201:220:222::2 }
    ip = {
      addr = bluedot.thun.net
    }
  }
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the `/etc/services` file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

Please note that if you use the DirAddresses directive, you must not use either a DirPort or a DirAddress directive in the same resource.

DirPort = <port-number> Specify the port (a positive integer) on which the Director daemon will listen for Bacula Console connections. This same port number must be specified in the Director resource of the Console configuration file. The default is 9101, so normally this directive need not be specified. This directive should not be used if you specify the DirAddresses (plural) directive.

DirAddress = <IP-Address> This directive is optional, but if it is specified, it will cause the Director server (for the Console program) to bind to the specified <IP-Address>, which is either a domain name or an IP address specified as a dotted quadruple in string or quoted string format. If this directive is not specified, the Director will bind to any available address (the default). Note, unlike the DirAddresses specification noted above, this directive only permits a single address to be specified. This directive should not be used if you specify a DirAddresses (plural) directive.

DirSourceAddress = <IP-Address> This record is optional, and if it is specified, it will cause the Director server (when initiating connections to a storage or file daemon) to source its connections from the specified address. Only a single IP address may be specified. If this record is not specified, the Director server will source its outgoing connections according to the system routing table (the default).

Events Retention = <time> The **Events Retention** directive defines the length of time that Bacula will keep events records in the Catalog database. When this time period expires, and if the user runs the `prune events` command, Bacula will prune (remove) Events records that are older than the specified period.

See the Configuration chapter of this manual for additional details of time specifications.

The default is 1 month.

Statistics Retention = <time> The **Statistics Retention** directive defines the length of time that Bacula will keep statistics job records in the Catalog database after the Job End time. (In `JobHistory` table) When this time period expires, and if the user runs the `prune stats` command, Bacula will prune (remove) Job records that are older than the specified period.

Theses statistics records aren't used for restore purpose, but mainly for capacity planning, billings, etc. See Statistics chapter for additional information.

See the Configuration chapter of this manual for additional details of time specifications.

The default is 5 years.

AutoPrune = <yes|no> Normally, pruning of Files from the Catalog is specified on a Client by Client basis in the `Client` resource with the **AutoPrune** directive. It is also possible to overwrite the Client settings in the `Pool` resource used by jobs, with the **AutoPrune**, **PruneFiles** and **PruneJobs** directives.

If this directive is specified (not normally) and the value is no, it will override the value specified in all the `Client` and the `Pool` resources. The default is yes.

If you set **AutoPrune**=no, pruning will not be done automatically, and your Catalog will grow in size each time you run a Job. Pruning affects only information in the catalog and not data stored in the backup archives (on Volumes). The `prune bconsole` command can be used to prune catalog records respecting the Client and/or the Pool **FileRetention**, **JobRetention** and **VolumeRetention** directives.

VerId = <string> where <string> is an identifier which can be used for support purpose. This string is displayed using the `version` command.

MaximumConsoleConnections = <number> where <number> is the maximum number of Console Connections that could run concurrently. The default is set to 20, but you may set it to a larger number.

MaximumReloadRequests = <number> Where <number> is the maximum number of `reload` command that can be queued while jobs are running. The default is set to 32 and is usually sufficient.

CommCompression = <yes|no> If the two Bacula components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled. In the case that the compression is not effective, Bacula turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports None in the Job report.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates

are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (bacula-fd.conf), Director resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the Client resource, director configuration file. We can have the below client configuration in bacula-dir.conf:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[1] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in bacula-fd.conf, Director resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
```

[1] https://en.wikipedia.org/wiki/X.509

```
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following is an example of a valid Director resource definition:

```
Director {
  Name = HeadMan
  WorkingDirectory = "$HOME/bacula/bin/working"
  Password = UA_password
  PidDirectory = "$HOME/bacula/bin/working"
  QueryFile = "$HOME/bacula/bin/query.sql"
  Messages = Standard
}
```

22.3 The Job Resource

The Job resource defines a Job (Backup, Restore, ...) that Bacula must perform. Each Job resource definition contains the name of a Client and a FileSet to backup, the Schedule for the Job, where the data are to be stored, and what media Pool can be used. In effect, each Job resource must specify What, Where, How, and When or FileSet, Storage, Backup/Restore/Level, and Schedule respectively. Note, the FileSet must be specified for a restore job for historical reasons, but it is no longer used.

Only a single type (Backup, Restore, ...) can be specified for any job. If you want to backup multiple FileSets on the same Client or multiple Clients, you must define a Job for each one.

Note, you define only a single Job to do the Full, Differential, and Incremental backups since the different backup levels are tied together by a unique Job name. Normally, you will have only one Job per Client, but if a client has a really huge number of files (more than several million), you might want to split it into to Jobs each with a different FileSet covering only part of the total files.

Multiple Storage daemons are not currently supported for Jobs, so if you do want to use multiple storage daemons, you will need to create a different Job and ensure that for each Job that the combination of Client and FileSet are unique. The Client and FileSet are what Bacula uses to restore a client, so if there are multiple Jobs with the same Client and FileSet or multiple Storage daemons that are used, the restore will not work. This problem can be resolved by defining multiple FileSet definitions (the names must be different, but the contents of the FileSets may be the same).

Job Start of the Job resource. At least one Job resource is required.

Name = <name> The Job name. This name can be specified on the `run` command in the console program to start a job. If the name contains spaces, it must be specified between quotes. It is generally a good idea to give your job the same name as the Client that it will backup. This permits easy identification of jobs.

When the job actually runs, the unique Job Name will consist of the name you specify here followed by the date and time the job was scheduled for execution. This directive is required.

Enabled = <yes|no> This directive allows you to enable or disable a `Job` resource. When the resource of the Job is disabled, the Job will no longer be scheduled and it will not be available in the list of Jobs to be run. To be able to use the Job you must `enable` it.

Tag = <string, string2, string3> The **Tag** directive specifies a list of tags to create when creating a new Job record. This directive is optional.

Type = <job-type> The **Type** directive specifies the Job type, which may be one of the following: Backup, Restore, Verify, or Admin. This directive is required. Within a particular Job Type, there are also Levels as discussed in the next item.

Backup Run a backup Job. Normally you will have at least one Backup job for each client you want to save. Normally, unless you turn off cataloging, most all the important statistics and data concerning files backed up will be placed in the catalog.

Restore Run a restore Job. Normally, you will specify only one Restore job which acts as a sort of prototype that you will modify using the console program in order to perform restores. Although certain basic information from a Restore job is saved in the catalog, it is very minimal compared to the information stored for a Backup job – for example, no File database entries are generated since no Files are saved.

Restore jobs cannot be automatically started by the scheduler as is the case for Backup, Verify and Admin jobs. To restore files, you must use the `restore` command in the console.

Verify Run a Verify Job. In general, Verify jobs permit you to compare the contents of the catalog to the file system, or to what was backed up. In addition, to verifying that a tape that was written can be read, you can also use Verify as a sort of tripwire intrusion detection.

Admin Run an Admin Job. Only Director's runscripts will be executed. The Client is not involved in an Admin job, so features such as **Client Run Before Job** are not available. Although an Admin job is recorded in the catalog, very little data is saved. An Admin job can be used to periodically run catalog pruning, if you do not want to do it at the end of each Backup Job.

Migration Run a Migration Job (similar to a backup job) that reads data that was previously backed up to a Volume and writes it to another Volume. (See 33 on page 411)

Copy Run a Copy Job that essentially creates two identical copies of the same backup. The Copy process is essentially identical to the Migration feature with the exception that the Job that is copied is left unchanged. (See 33 on page 411)

Level = <job-level> The Level directive specifies the default Job level to be run. Each different Job Type (Backup, Restore, ...) has a different set of Levels that can be specified. The Level is normally overridden by a different value that is specified in the `Schedule` resource. This directive is not required, but must be specified either by a **Level** directive or as an override specified in the `Schedule` resource.

For a Backup Job, the Level may be one of the following:

Full When the **Level** is set to Full all files in the FileSet whether or not they have changed will be backed up.

Incremental When the Level is set to Incremental all files specified in the FileSet that have changed since the last successful backup of the the same Job using the same FileSet and Client, will be backed up. If the Director cannot find a previous valid Full backup then the job will be upgraded into a Full backup. When the Director looks for a valid backup record in the catalog database, it looks for a previous Job with:

- The same Job name.
- The same Client name.
- The same FileSet (any change to the definition of the FileSet such as adding or deleting a file in the Include or Exclude sections constitutes a different FileSet.
- The Job was a Full, Differential, or Incremental backup.
- The Job terminated normally (i.e. did not fail or was not canceled).
- The Job started no longer ago than **Max Full Interval**.

If all the above conditions do not hold, the Director will upgrade the Incremental to a Full save. Otherwise, the Incremental backup will be performed as requested.

The File daemon (Client) decides which files to backup for an Incremental backup by comparing start time of the prior Job (Full, Differential, or Incremental) against the time each file was last "modified" (`st_mtime`) and the time its attributes were last "changed"(`st_ctime`). If the file was modified or its attributes changed on or after this start time, it will then be backed up.

Some virus scanning software may change `st_ctime` while doing the scan. For example, if the virus scanning program attempts to reset the access time (`st_atime`), which Bacula does not use, it will cause `st_ctime` to change and hence Bacula will backup the file during an Incremental or Differential backup. In the case of Sophos

virus scanning, you can prevent it from resetting the access time (st_atime) and hence changing st_ctime by using the --**no-reset-atime** option. For other software, please see their manual.

When Bacula does an Incremental backup, all modified files that are still on the system are backed up. However, any file that has been deleted since the last Full backup remains in the Bacula catalog, which means that if between a Full save and the time you do a restore, some files are deleted, those deleted files will also be restored. The deleted files will no longer appear in the catalog after doing another Full save.

In addition, if you move a directory rather than copy it, the files in it do not have their modification time (st_mtime) or their attribute change time (st_ctime) changed. As a consequence, those files will probably not be backed up by an Incremental or Differential backup which depend solely on these time stamps. If you move a directory, and wish it to be properly backed up, it is generally preferable to copy it, then delete the original.

However, to manage deleted files or directories changes in the catalog during an Incremental backup you can use accurate mode. This is quite memory consuming process. See Accurate mode for more details.

Differential When the Level is set to Differential all files specified in the FileSet that have changed since the last successful Full backup of the same Job will be backed up. If the Director cannot find a valid previous Full backup for the same Job, FileSet, and Client, backup, then the Differential job will be upgraded into a Full backup. When the Director looks for a valid Full backup record in the catalog database, it looks for a previous Job with:

- The same Job name.
- The same Client name.
- The same FileSet (any change to the definition of the FileSet such as adding or deleting a file in the Include or Exclude sections constitutes a different FileSet.
- The Job was a FULL backup.
- The Job terminated normally (i.e. did not fail or was not canceled).
- The Job started no longer ago than **Max Full Interval**.

If all the above conditions do not hold, the Director will upgrade the Differential to a Full save. Otherwise, the Differential backup will be performed as requested.

The File daemon (Client) decides which files to backup for a differential backup by comparing the start time of the prior Full backup Job against the time each file was last "modified" (st_mtime) and the time its attributes were last "changed" (st_ctime). If the file was modified or its attributes were changed on or after this start time, it will then be backed up. The start time used is displayed after the **Since** on the Job report. In rare cases, using the start time of the prior backup may cause some files to be backed up twice, but it ensures that no change is missed. As with the Incremental option, you should ensure that the clocks on your server and client are synchronized or as close as possible to avoid the possibility of a file being skipped. Note, on versions 1.33 or greater Bacula automatically makes the necessary adjustments to the time between the server and the client so that the times Bacula uses are synchronized.

When Bacula does a Differential backup, all modified files that are still on the system are backed up. However, any file that has been deleted since the last Full backup remains in the Bacula catalog, which means that if between a Full save and the time you do a restore, some files are deleted, those deleted files will also be restored. The deleted files will no longer appear in the catalog after doing another Full save. However, to remove deleted files from the catalog during a Differential backup is quite a time consuming process and not currently implemented in Bacula. It is, however, a planned future feature.

As noted above, if you move a directory rather than copy it, the files in it do not have their modification time (st_mtime) or their attribute change time (st_ctime) changed. As a consequence, those files will probably not be backed up by an Incremental or Differential backup which depend solely on these time stamps. If you move

a directory, and wish it to be properly backed up, it is generally preferable to copy it, then delete the original. Alternatively, you can move the directory, then use the touch program to update the timestamps.

However, to manage deleted files or directories changes in the catalog during an Differential backup you can use accurate mode. This is quite memory consuming process. See Accurate mode for more details.

Every once and a while, someone asks why we need Differential backups as long as Incremental backups pickup all changed files. There are possibly many answers to this question, but the one that is the most important for me is that a Differential backup effectively merges all the Incremental and Differential backups since the last Full backup into a single Differential backup. This has two effects:

1. It gives some redundancy since the old backups could be used if the merged backup cannot be read.
2. More importantly, it reduces the number of Volumes that are needed to do a restore effectively eliminating the need to read all the volumes on which the preceding Incremental and Differential backups since the last Full are done.

VirtualFull When the backup Level is set to VirtualFull, Bacula will consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

Bacula's virtual backup feature is often called Synthetic Backup or Consolidation in other backup products.

For a Restore Job, no level needs to be specified.

For a Verify Job, the Level may be one of the following:

InitCatalog does a scan of the specified **FileSet** and stores the file attributes in the Catalog database. Since no file data is saved, you might ask why you would want to do this. It turns out to be a very simple and easy way to have a **Tripwire** like feature using Bacula. In other words, it allows you to save the state of a set of files defined by the `FileSet` and later check to see if those files have been modified or deleted and if any new files have been added. This can be used to detect system intrusion. Typically you would specify a `FileSet` that contains the set of system files that should not change (e.g. /sbin, /boot, /lib, /bin, ...). Normally, you run the InitCatalog level verify one time when your system is first setup, and then once again after each modification (upgrade) to your system. Thereafter, when your want to check the state of your system files, you use a Verify **level**=Catalog. This compares the results of your InitCatalog with the current state of the files.

Catalog Compares the current state of the files against the state previously saved during an InitCatalog. Any discrepancies are reported. The items reported are determined by the Verify options specified on the **Include** directive in the specified `FileSet` (see the `FileSet` resource below for more details). Typically this command will be run once a day (or night) to check for any changes to your system files.

Please note! If you run two Verify Catalog jobs on the same client at the same time, the results will certainly be incorrect. This is because Verify Catalog modifies the Catalog database while running in order to track new files.

Data Read back the data stored on volumes and check data attributes such as size and the checksum of all the files.

To run the Verify job, it is possible to use the "jobid" parameter of the "run" command.

Please note, the current Verify Data implementation requires specifying the correct Storage resource in the Verify job. The Storage resource can be changed with the bconsole command line and with the menu.

It is also possible to use the `accurate` option to check catalog records at the same time. When using a Verify job with `level=Data` and `accurate=yes` can replace the `level=VolumeToCatalog` option.

To run a Verify Job with the accurate option, it is possible to set the option in the Job definition or set use the accurate=yes on the command line.

```
* run job=VerifyData level=Data jobid=10 accurate=yes
```

VolumeToCatalog This level causes Bacula to read the file attribute data written to the Volume from the last backup Job for the job specified on the **VerifyJob** directive. The file attribute data are compared to the values saved in the Catalogind database and any differences are reported. This is similar to the DiskToCatalog level except that instead of comparing the disk file attributes to the catalog database, the attribute data written to the Volume is read and compared to the catalog database. Although the attribute data including the signatures (MD5 or SHA1) are compared, the actual file data is not compared (it is not in the catalog).

Please note! If you run two Verify VolumeToCatalog jobs on the same client at the same time, the results will certainly be incorrect. This is because the Verify VolumeToCatalog modifies the Catalog database while running.

DiskToCatalog This level causes Bacula to read the files as they currently are on disk, and to compare the current file attributes with the attributes saved in the catalog from the last backup for the job specified on the **VerifyJob** directive. This level differs from the VolumeToCatalog level described above by the fact that it doesn't compare against a previous Verify job but against a previous backup. When you run this level, you must supply the verify options on your Include statements. Those options determine what attribute fields are compared.

This command can be very useful if you have disk problems because it will compare the current state of your disk against the last successful backup, which may be several jobs.

Note, the current implementation (1.32c) does not identify files that have been deleted.

Accurate = <yes|no> In accurate mode, the File daemon knowns exactly which files were present after the last backup. So it is able to handle deleted or renamed files.

When restoring a FileSet for a specified date (including "most recent"), Bacula is able to restore exactly the files and directories that existed at the time of the last backup prior to that date including ensuring that deleted files are actually deleted, and renamed directories are restored properly.

In this mode, the File daemon must keep data concerning all files in memory. So If you do not have sufficient memory, the backup may either be terribly slow or fail.

For 500.000 files (a typical desktop linux system), it will require approximately 64 Megabytes of RAM on your File daemon to hold the required information.

Verify Job = <Job-Resource-Name> If you run a verify job without this directive, the last job run will be compared with the catalog, which means that you must immediately follow a backup by a verify command. If you specify a Verify Job Bacula will find the last job with that name that ran. This permits you to run all your backups, then run Verify jobs on those that you wish to be verified (most often a VolumeToCatalog) so that the tape just written is re-read.

JobDefs = <JobDefs-Resource-Name> If a <JobDefs-Resource-Name> is specified, all the values contained in the named JobDefs resource will be used as the defaults for the current Job. Any value that you explicitly define in the current Job resource, will override any defaults specified in the JobDefs resource. The use of this directive permits writing much more compact Job resources where the bulk of the directives are defined in one or more JobDefs. This is particularly useful if you have many similar Jobs but with minor variations such as different Clients. A simple example of the use of JobDefs is provided in the default bacula-dir.conf file.

Bootstrap = <bootstrap-file> The **Bootstrap** directive specifies a bootstrap file that, if provided, will be used during Restore Jobs and is ignored in other Job types. The <bootstrap-file> contains the list of tapes to be used in a Restore Job as well as which files are to be restored. Specification of this directive is optional, and if specified, it is used only for a

restore job. In addition, when running a Restore job from the console, this value can be changed.

If you use the `restore` command in the `bconsole` program, to start a Restore job, the <bootstrap-file> will be created automatically from the files you select to be restored.

For additional details of the **bootstrap** directive, please see Restoring Files with the Bootstrap File chapter of this manual.

Write Bootstrap = <bootstrap-file-specification> The `writebootstrap` directive specifies a file name where Bacula will write a **bootstrap** file for each Backup job run. This directive applies only to Backup Jobs. If the Backup job is a Full save, Bacula will erase any current contents of the specified file before writing the bootstrap records. If the Job is an Incremental or Differential save, Bacula will append the current bootstrap record to the end of the file.

Using this feature, permits you to constantly have a bootstrap file that can recover the current state of your system. Normally, the file specified should be a mounted drive on another machine, so that if your hard disk is lost, you will immediately have a bootstrap record available. Alternatively, you should copy the bootstrap file to another machine after it is updated. Note, it is a good idea to write a separate bootstrap file for each Job backed up including the job that backs up your catalog database.

If the <bootstrap-file-specification> begins with a vertical bar (|), Bacula will use the specification as the name of a program to which it will pipe the bootstrap record. It could for example be a shell script that emails you the bootstrap record.

On versions 1.39.22 or greater, before opening the file or executing the specified command, Bacula performs character substitution like in RunScript directive. To automatically manage your bootstrap files, you can use this in your `JobDefs` resources:

```
JobDefs {
   Write Bootstrap = "%c_%n.bsr"
   ...
}
```

For more details on using this file, please see the chapter entitled The Bootstrap File of this manual.

Client = <client-resource-name> The Client directive specifies the Client (File daemon) that will be used in the current Job. Only a single Client may be specified in any one Job. The Client runs on the machine to be backed up, and sends the requested files to the Storage daemon for backup, or receives them when restoring. For additional details, see the Client Resource section of this chapter. This directive is required.

FileSet = <FileSet-resource-name> The FileSet directive specifies the FileSet that will be used in the current Job. The FileSet specifies which directories (or files) are to be backed up, and what options to use (e.g. compression, ...). Only a single FileSet resource may be specified in any one Job. For additional details, see the FileSet Resource section of this chapter. This directive is required.

Base = <job-resource-name, ...> The Base directive permits to specify the list of jobs that will be used during Full backup as base. This directive is optional. See the Base Job chapter for more information.

Messages = <messages-resource-name> The Messages directive defines what Messages resource should be used for this job, and thus how and where the various messages are to be delivered. For example, you can direct some messages to a log file, and others can be sent by email. For additional details, see the Messages Resource Chapter of this manual. This directive is required.

Snapshot Retention = <time-period-specification> The Snapshot Retention directive defines the length of time that Bacula will keep Snapshots in the Catalog database and on the Client after the Snapshot creation. When this time period expires, and if using the `snapshot prune` command, Bacula will prune (remove) Snapshot records that are older than the specified Snapshot Retention period and will contact the FileDaemon to delete Snapshots from the system.

The Snapshot retention period is specified as seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 0 seconds, Snapshots are deleted at the end of the backup. The Job **SnapshotRetention** directive overwrites the Client **SnapshotRetention** directive.

Pool = <pool-resource-name> The Pool directive defines the pool of Volumes where your data can be backed up. Many Bacula installations will use only the Default pool. However, if you want to specify a different set of Volumes for different Clients or different Jobs, you will probably want to use Pools. For additional details, see the Pool Resource section of this chapter. This directive is required.

Full Backup Pool = <pool-resource-name> The **Full Backup Pool** specifies a Pool to be used for Full backups. It will override any Pool specification during a Full backup. This directive is optional.

Differential Backup Pool = <pool-resource-name> The **Differential Backup Pool** specifies a Pool to be used for Differential backups. It will override any Pool specification during a Differential backup. This directive is optional.

Incremental Backup Pool = <pool-resource-name> The **Incremental Backup Pool** specifies a Pool to be used for Incremental backups. It will override any Pool specification during an Incremental backup. This directive is optional.

VirtualFull Backup Pool = <pool-resource-name> The **VirtualFull Backup Pool** specifies a Pool to be used for VirtualFull backups. It will override any Pool specification during an VirtualFull backup. This directive is optional.

BackupsToKeep = <number> When this directive is present during a Virtual Full (it is ignored for other Job types), it will look for a Full backup that has more subsequent backups than the value specified. In the example below, the Job will simply terminate unless there is a Full back followed by at least 31 backups of either level Differential or Incremental.

```
Job {
  Name = "VFull"
  Type = Backup
  Level = VirtualFull
  Client = "my-fd"
  File Set = "FullSet"
  Accurate = Yes
  Backups To Keep = 30
}
```

Assuming that the last Full backup is followed by 32 Incremental backups, a Virtual Full will be run that consolidates the Full with the first two Incrementals that were run after the Full. The result is that you will end up with a Full followed by 30 Incremental backups.

DeleteConsolidatedJobs = <yes/no> If set to yes, it will cause any old Job that is consolidated during a Virtual Full to be deleted. In the example above we saw that a Full plus one other job (either an Incremental or Differential) were consolidated into a new Full backup. The original Full plus the other Job consolidated will be deleted. The default value is no.

Schedule = <schedule-name> The Schedule directive defines what schedule is to be used for the Job. The schedule in turn determines when the Job will be automatically started and what Job level (i.e. Full, Incremental, ...) is to be run. This directive is optional, and if left out, the Job can only be started manually using the Console program. Although you may specify only a single Schedule resource for any one job, the Schedule resource may contain multiple **Run** directives, which allow you to run the Job at many different times, and each **Run** directive permits overriding the default Job Level Pool, Storage, and Messages resources. This gives considerable flexibility in what can be done with a single Job. For additional details, see the Schedule Resource chapter of this manual.

Storage = <storage-resource-name> The Storage directive defines the name of the storage services where you want to backup the FileSet data. For additional details, see the Storage Resource Chapter of this manual. The Storage resource may also be specified in the Job's

Pool resource, in which case the value in the Pool resource overrides any value in the Job. This Storage resource definition is not required by either the Job resource or in the Pool, but it must be specified in one or the other, if not an error will result. Storage can be either specified by an single item or it can be specified as a comma separated list of storages to use according to StorageGroupPolicy. If some number of first storage daemons on the list are unavailable due to network problems, broken or unreachable for some other reason, Bacula will take first available one from the list (which is sorted according to the policy used) which is network reachable and healthy.

StorageGroupPolicy = <Storage Group Policy Name> Storage Group Policy determines how Storage resources (from the 'Storage' directive) are being choosen from the Storage list. If no StoragePolicy is specified Bacula always tries to use first available Storage from the provided list. Currently supported policies are:

ListedOrder - This is the default policy, which uses first available storage from the list provided

LeastUsed - This policy scans all storage daemons from the list and chooses the one with the least number of jobs being currently run

Max Start Delay = <time> The time specifies the maximum delay between the scheduled time and the actual start time for the Job. For example, a job can be scheduled to run at 1:00am, but because other jobs are running, it may wait to run. If the delay is set to 3600 (one hour) and the job has not begun to run by 2:00am, the job will be canceled. This can be useful, for example, to prevent jobs from running during day time hours. The default is 0 which indicates no limit.

Max Run Time = <time> The time specifies the maximum allowed time that a job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

By default, the the watchdog thread will kill any Job that has run more than 200 days. The maximum watchdog timeout is independent of MaxRunTime and cannot be changed.

Incremental|Differential Max Wait Time = <time> Theses directives have been deprecated in favor of **Incremental|Differential Max Run Time** since bacula 2.3.18.

Incremental Max Run Time = <time> The time specifies the maximum allowed time that an Incremental backup job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

Differential Max Run Time = <time> The time specifies the maximum allowed time that a Differential backup job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

Max Run Sched Time = <time> The time specifies the maximum allowed time that a job may run, counted from when the job was scheduled. This can be useful to prevent jobs from running during working hours. We can see it like `Max Start Delay + Max Run Time`.

Max Wait Time = <time> The time specifies the maximum allowed time that a job may block waiting for a resource (such as waiting for a tape to be mounted, or waiting for the storage or file daemons to perform their duties), counted from the when the job starts, (**not** necessarily the same as when the job was scheduled). This directive works as expected since bacula 2.3.18.

Maximum Spawned Jobs = <nb> The Job resource now permits specifying a number of **Maximum Spawn Jobs**. The default is 600. This directive can be useful if you have big hardware and you do a lot of Migration/Copy jobs which start at the same time.

Maximum Bandwidth = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes that a job may use. You may specify the following speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfer with the value set with the Directive.

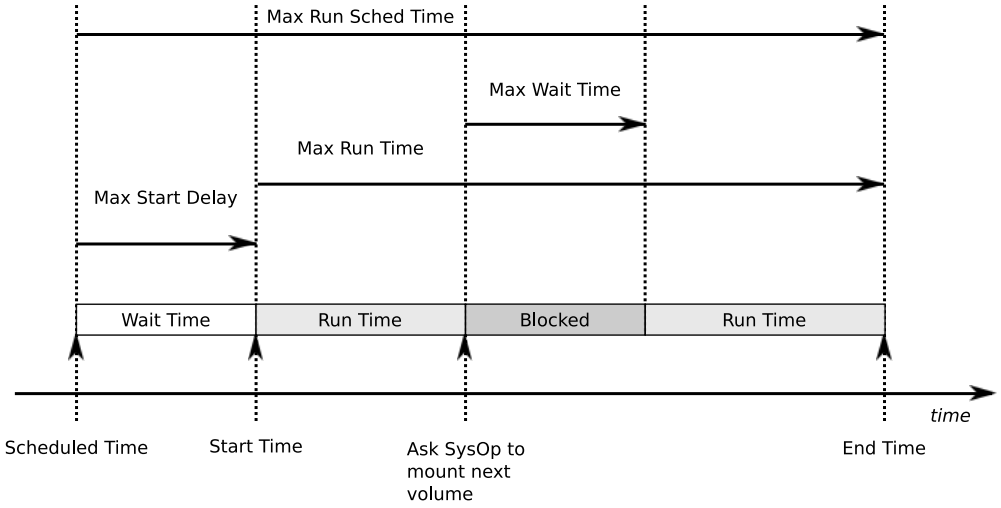


Figure 22.1: Job time control directives

Max Full Interval = <time> The time specifies the maximum allowed age (counting from start time) of the most recent successful Full backup that is required in order to run Incremental or Differential backup jobs. If the most recent Full backup is older than this interval, Incremental and Differential backups will be upgraded to Full backups automatically. If this directive is not present, or specified as 0, then the age of the previous Full backup is not considered.

Max VirtualFull Interval = <time> The time specifies the maximum allowed age (counting from start time) of the most recent successful Full backup that is required in order to run Incremental, Differential or Full backup jobs. If the most recent Full backup is older than this interval, Incremental, Differential and Full backups will be converted to a VirtualFull backup automatically. If this directive is not present, or specified as 0, then the age of the previous Full backup is not considered.

Please note that a VirtualFull job is not a real backup job. A VirtualFull will merge exiting jobs to create a new virtual Full job in the catalog and will copy the exiting data to new volumes.

The Client is not used in a VirtualFull job, so when using this directive, the Job that was supposed to run and save recently modified data on the Client will not run. Only the next regular Job defined in the Schedule will backup the data. It will not be possible to restore the data that was modified on the Client between the last Incremental/Differential and the VirtualFull.

Prefer Mounted Volumes = <yes|no> If the Prefer Mounted Volumes directive is set to yes (default yes), the Storage daemon is requested to select either an Autochanger or a drive with a valid Volume already mounted in preference to a drive that is not ready. This means that all jobs will attempt to append to the same Volume (providing the Volume is appropriate – right Pool, ... for that job), unless you are using multiple pools. If no drive with a suitable Volume is available, it will select the first available drive. Note, any Volume that has been requested to be mounted, will be considered valid as a mounted volume by another job. This if multiple jobs start at the same time and they all prefer mounted volumes, the first job will request the mount, and the other jobs will use the same volume.

If the directive is set to no, the Storage daemon will prefer finding an unused drive, otherwise, each job started will append to the same Volume (assuming the Pool is the same for all jobs). Setting Prefer Mounted Volumes to no can be useful for those sites with multiple drive autochangers that prefer to maximize backup throughput at the expense of using additional drives and Volumes. This means that the job will prefer to use an unused drive rather than use a drive that is already in use.

Despite the above, we recommend against setting this directive to no since it tends to add a lot of swapping of Volumes between the different drives and can easily lead to deadlock situations in the Storage daemon. We will accept bug reports against it, but we cannot guarantee that we will be able to fix the problem in a reasonable time.

A better alternative for using multiple drives is to use multiple pools so that Bacula will be forced to mount Volumes from those Pools on different drives.

Prune Jobs = <yes|no> Normally, pruning of Jobs from the Catalog is specified on a Client by Client basis in the Client resource with the **AutoPrune** directive. If this directive is specified (not normally) and the value is yes, it will override the value specified in the Client resource. The default is no.

Prune Files = <yes|no> Normally, pruning of Files from the Catalog is specified on a Client by Client basis in the Client resource with the **AutoPrune** directive. If this directive is specified (not normally) and the value is yes, it will override the value specified in the Client resource. The default is no.

Prune Volumes = <yes|no> Normally, pruning of Volumes from the Catalog is specified on a Pool by Pool basis in the Pool resource with the **AutoPrune** directive. Note, this is different from File and Job pruning which is done on a Client by Client basis. If this directive is specified (not normally) and the value is yes, it will override the value specified in the Pool resource. The default is no.

RunScript {<body-of-runscript>} The **RunScript** directive behaves like a resource in that it requires opening and closing braces around a number of directives that make up the body of the runscript.

The specified **Command** (see below for details) is run as an external program prior or after the current Job. This is optional. By default, the program is executed on the Client side like in `ClientRunXXXJob`.

Console options are special commands that are sent to the director instead of the OS. At this time, console command ouputs are redirected to log with the jobid 0.

You can use following console command : `delete`, `disable`, `enable`, `estimate`, `list`, `llist`, `memory`, `prune`, `purge`, `reload`, `status`, `setdebug`, `show`, `time`, `trace`, `update`, `version`, `.client`, `.jobs`, `.pool`, `.storage`. See console chapter for more information. You need to specify needed information on command line, nothing will be prompted. Example :

```
Console = "prune files client=%c"
Console = "update stats age=3"
```

You can specify more than one Command/Console option per RunScript.

You can use following options may be specified in the body of the runscript:

Table 22.1: Options for Run Script

Options	Value	Default	Information
Runs On Success	Yes / No	Yes	Run command if JobStatus is successful
Runs On Failure	Yes / No	No	Run command if JobStatus isn't successful
Runs On Client	Yes / No	Yes	Run command on client[2]

Continues on the following page

[2]Scripts will run on Client only with Jobs that use a Client. (Backup, Restore, some Verify jobs). For other Jobs (Copy, Migration, Admin, ...) RunsOnClient should be set to No.

[]Cont.

Runs When	Before After Always *AfterVSS*	Never	When run commands
Fail Job On Error	Yes/No	Yes	Fail job if script returns something different from 0
Command			Path to your script
Console			Console command

Any output sent by the command to standard output will be included in the Bacula job report. The command string must be a valid program name or name of a shell script.

In addition, the command string is parsed then fed to the OS, which means that the path will be searched to execute your specified command, but there is no shell interpretation, as a consequence, if you invoke complicated commands or want any shell features such as redirection or piping, you must call a shell script and do it inside that script.

Before submitting the specified command to the operating system, Bacula performs character substitution of the following characters:

```
%% = %
%b = Job Bytes
%c = Client's name
%C = If the job is a Cloned job (Only on director side)
%d = Daemon's name (Such as host-dir or host-fd)
%D = Director's name (Also valid on file daemon)
%e = Job Exit Status
%E = Non-fatal Job Errors
%f = Job FileSet (Only on director side)
%F = Job Files
%h = Client address
%i = JobId
%I = Migration/Copy JobId (Only in Copy/Migrate Jobs)
%j = Unique Job id
%l = Job Level
%n = Job name
%o = Job Priority
%p = Pool name (Only on director side)
%P = Current PID process
%R = Read Bytes
%s = Since time
%S = Previous Job name (Only on file daemon side)
%t = Job type (Backup, ...)
%v = Volume name (Only on director side)
%w = Storage name (Only on director side)
%x = Spooling enabled? ("yes" or "no")
```

Some character substitutions are not available in all situations. The Job Exit Status code %e edits the following values:

- OK
- Error
- Fatal Error
- Canceled
- Differences
- Unknown term code

Thus if you edit it on a command line, you will need to enclose it within some sort of quotes.

You can use these following shortcuts:

Table 22.2: RunScript shortcuts

Keyword	Runs On Success	Runs On Failure	FailJob On Error	Runs On Client	Runs When
Run Before Job			Yes	No	Before
Run After Job	Yes	No		No	After
Run After Failed Job	No	Yes		No	After
Client Run Before Job			Yes	Yes	Before
Client Run After Job	Yes	No		Yes	After

Examples:

```
RunScript {
    RunsWhen = Before
    FailJobOnError = No
    Command = "/etc/init.d/apache stop"
}

RunScript {
    RunsWhen = After
    RunsOnFailure = yes
    Command = "/etc/init.d/apache start"
}
```

Notes about ClientRunBeforeJob

For compatibility reasons, with this shortcut, the command is executed directly when the client recieve it. And if the command is in error, other remote runscripts will be discarded. To be sure that all commands will be sent and executed, you have to use RunScript syntax.

Special Shell Considerations

A "Command =" can be one of:

- The full path to an executable program
- The name of an executable program that can be found in the $PATH
- A *complex* shell command in the form of: "sh -c \"your commands go here\""

Special Windows Considerations

You can run scripts just after snapshots initializations with AfterVSS keyword.

In addition, for a Windows client, please take note that you must ensure a correct path to your script. The script or program can be a `.com`, `.exe` or a `.bat` file. If you just put the program name in then Bacula will search using the same rules that `cmd.exe` uses (current directory, Bacula bin directory, and `PATH`). It will even try the different extensions in the same order as `cmd.exe`. The command can be anything that `cmd.exe` or `command.com` will recognize as an executable file.

However, if you have slashes in the program name then Bacula figures you are fully specifying the name, so you must also explicitly add the three character extension.

The command is run in a Win32 environment, so Unix like commands will not work unless you have installed and properly configured Cygwin in addition to and separately from Bacula.

The System `%Path%` will be searched for the command. (under the environment variable dialog you have have both System Environment and User Environment, we believe that only the System environment will be available to `bacula-fd`, if it is running as a service.)

System environment variables can be referenced with `%var%` and used as either part of the command name or arguments.

So if you have a script in the Bacula `\bin` directory then the following lines should work fine:

```
        Client Run Before Job = systemstate
or
        Client Run Before Job = systemstate.bat
or
        Client Run Before Job = "systemstate"
or
        Client Run Before Job = "systemstate.bat"
or
        ClientRunBeforeJob = "\"C:/Program Files/Bacula/systemstate.bat\""
```

The outer set of quotes is removed when the configuration file is parsed. You need to escape the inner quotes so that they are there when the code that parses the command line for execution runs so it can tell what the program name is.

```
ClientRunBeforeJob = "\"C:/Program Files/Software
     Vendor/Executable\" /arg1 /arg2 \"foo bar\""
```

The special characters

```
&<>()@^|
```

will need to be quoted, if they are part of a filename or argument.

If someone is logged in, a blank "command" window running the commands will be present during the execution of the command.

Some Suggestions from Phil Stracchino for running on Win32 machines with the native Win32 File daemon:

1. You might want the ClientRunBeforeJob directive to specify a `.bat` file which runs the actual client-side commands, rather than trying to run (for example) `regedit /e` directly.
2. The batch file should explicitly "`exit 0`" on successful completion.
3. The path to the batch file should be specified in Unix form:

   ```
   ClientRunBeforeJob = "c:/bacula/bin/systemstate.bat"
   ```

   rather than DOS/Windows form:

   ```
   ClientRunBeforeJob = "c:\bacula\bin\systemstate.bat"  # INCORRECT
   ```

For Win32, please note that there are certain limitations:

```
ClientRunBeforeJob = "C:/Program Files/Bacula/bin/pre-exec.bat"
```

Lines like the above do not work because there are limitations of `cmd.exe` that is used to execute the command. Bacula prefixes the string you supply with `cmd.exe /c` . To test that your command works you should type `cmd /c "C:/Program Files/test.exe"` at a cmd prompt and see what happens. Once the command is correct insert a backslash (\) before each double quote ("), and then put quotes around the whole thing when putting it in the director's configuration file. You either need to have only one set of quotes or else use the short name and don't put quotes around the command path.

Below is the output from cmd's help as it relates to the command line passed to the `/c` option.

If `/C` or `/K` is specified, then the remainder of the command line after the switch is processed as a command line, where the following logic is used to process quote (") characters:

1. If all of the following conditions are met, then quote characters on the command line are preserved:
   - no `/S` switch.
   - exactly two quote characters.
   - no special characters between the two quote characters, where special is one of:

     ```
     &<>()@^|
     ```

   - there are one or more whitespace characters between the the two quote characters.

- the string between the two quote characters is the name of an executable file.

2. Otherwise, old behavior is to see if the first character is a quote character and if so, strip the leading character and remove the last quote character on the command line, preserving any text after the last quote character.

The following example of the use of the Client Run Before Job directive was submitted by a user:

You could write a shell script to back up a DB2 database to a FIFO. The shell script is:

```
#!/bin/sh
# ===== backupdb.sh
DIR=/u01/mercuryd

mkfifo $DIR/dbpipe
db2 BACKUP DATABASE mercuryd TO $DIR/dbpipe WITHOUT PROMPTING &
sleep 1
```

The following line in the Job resource in the bacula-dir.conf file:

```
Client Run Before Job = "su - mercuryd -c \"/u01/mercuryd/backupdb.sh '%t' '%l'\""
```

When the job is run, you will get messages from the output of the script stating that the backup has started. Even though the command being run is backgrounded with &, the job will block until the db2 BACKUP DATABASE command, thus the backup stalls.

To remedy this situation, the "db2 BACKUP DATABASE" line should be changed to the following:

```
db2 BACKUP DATABASE mercuryd TO $DIR/dbpipe WITHOUT PROMPTING > $DIR/backup.log 2>&1 < /dev/null &
```

It is important to redirect the input and outputs of a backgrounded command to /dev/null to prevent the script from blocking.

Run Before Job = <command> The specified <command> is run as an external program prior to running the current Job. This directive is not required, but if it is defined, and if the exit code of the program run is non-zero, the current Bacula job will be canceled.

```
Run Before Job = "echo test"
```

it's equivalent to :

```
RunScript {
 Command = "echo test"
 RunsOnClient = No
 RunsWhen = Before
}
```

Lutz Kittler has pointed out that using the RunBeforeJob directive can be a simple way to modify your schedules during a holiday. For example, suppose that you normally do Full backups on Fridays, but Thursday and Friday are holidays. To avoid having to change tapes between Thursday and Friday when no one is in the office, you can create a RunBeforeJob that returns a non-zero status on Thursday and zero on all other days. That way, the Thursday job will not run, and on Friday the tape you inserted on Wednesday before leaving will be used.

Run After Job = <Command> The specified <Command> is run as an external program if the current job terminates normally (without error or without being canceled). This directive is not required. If the exit code of the program run is non-zero, Bacula will print a warning message. Before submitting the specified command to the operating system, Bacula performs character substitution as described above for the **RunScript** directive.

An example of the use of this directive is given in the **Tips** chapter (chapter 2.4 page 13) of the Bacula Community Edition Problems Resolution Guide.

See the **Run After Failed Job** if you want to run a script after the job has terminated with any non-normal status.

Run After Failed Job = <Command> The specified <Command> is run as an external program after the current job terminates with any error status. This directive is not required. The command string must be a valid program name or name of a shell script. If the exit code of the program run is non-zero, Bacula will print a warning message. Before submitting the specified command to the operating system, Bacula performs character substitution as described above for the **RunScript** directive. Note, if you wish that your script will run regardless of the exit status of the Job, you can use this :

```
RunScript {
  Command = "echo test"
  RunsWhen = After
  RunsOnFailure = yes
  RunsOnClient  = no
  RunsOnSuccess = yes    # default, you can drop this line
}
```

An example of the use of this directive is given in the **Tips** chapter (chapter 2.4 page 13) of the Bacula Community Edition Problems Resolution Guide.

Client Run Before Job = <Command> This directive is the same as **Run Before Job** except that the program is run on the client machine. The same restrictions apply to Unix systems as noted above for the **RunScript**. **ClientRunBeforeJob** can be used with Backup and Restore jobs.

Client Run After Job = <Command> The specified <Command> is run on the client machine as soon as data spooling is complete in order to allow restarting applications on the client as soon as possible. **ClientRunBeforeJob** can be used with Backup and Restore jobs.

Note, please see the notes above in **RunScript** concerning Windows clients.

Rerun Failed Levels = <yes|no> If this directive is set to yes (default no), and Bacula detects that a previous job at a higher level (i.e. Full or Differential) has failed, the current job level will be upgraded to the higher level. This is particularly useful for Laptops where they may often be unreachable, and if a prior Full save has failed, you wish the very next backup to be a Full save rather than whatever level it is started as.

There are several points that must be taken into account when using this directive: first, a failed job is defined as one that has not terminated normally, which includes any running job of the same name (you need to ensure that two jobs of the same name do not run simultaneously); secondly, the **Ignore FileSet Changes** directive is not considered when checking for failed levels, which means that any FileSet change will trigger a rerun.

Spool Data = <yes|no> If this directive is set to yes (default no), the Storage daemon will be requested to spool the data for this Job to disk rather than write it directly to the Volume (normally a tape).

Thus the data is written in large blocks to the Volume rather than small blocks. This directive is particularly useful when running multiple simultaneous backups to tape. Once all the data arrives or the spool files' maximum sizes are reached, the data will be despooled and written to tape.

Spooling data prevents interleaving date from several job and reduces or eliminates tape drive stop and start commonly known as "shoe-shine".

We don't recommend using this option if you are writing to a disk file using this option will probably just slow down the backup jobs.

NOTE: When this directive is set to yes, Spool Attributes is also automatically set to yes.

Spool Attributes = <yes|no> The default is set to yes, the Storage daemon will buffer the File attributes and Storage coordinates to a temporary file in the Working Directory, then when writing the Job data to the tape is completed, the attributes and storage coordinates will be sent to the Director. If set to no the File attributes are sent by the Storage daemon to the Director as they are stored on tape.

NOTE: When Spool Data is set to yes, Spool Attributes is also automatically set to yes.

SpoolSize=*bytes* where the bytes specify the maximum spool size for this job. The default is take from Device Maximum Spool Size limit. This directive is available only in Bacula version 2.3.5 or later.

Where = <directory> This directive applies only to a Restore job and specifies a prefix to the directory name of all files being restored. This permits files to be restored in a different location from which they were saved. If **Where** is not specified or is set to slash (/), the files will be restored to their original location. By default, we have set **Where** in the example configuration files to be `/tmp/bacula-restores`. This is to prevent accidental overwriting of your files.

Add Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to the directory name of all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

Add Suffix = <extention> This directive applies only to a Restore job and specifies a suffix to all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

Using `Add Suffix=.old`, `/etc/passwd` will be restored to `/etc/passwsd.old`

Strip Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to remove from the directory name of all files being restored. This will use the File Relocation feature implemented in Bacula 2.1.8 or later.

Using `Strip Prefix=/etc`, `/etc/passwd` will be restored to `/passwd`

Under Windows, if you want to restore `c:/files` to `d:/files`, you can use :

```
Strip Prefix = c:
Add Prefix = d:
```

RegexWhere = <expressions> This directive applies only to a Restore job and specifies a regex filename manipulation of all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

For more informations about how use this option, see this.

Replace = <replace-option> This directive applies only to a Restore job and specifies what happens when Bacula wants to restore a file or directory that already exists. You have the following options for <replace-option>:

RestoreClient = <client-resource-name> The RestoreClient directive specifies the default Client (File Daemon) that will be used with the restore job. If this directive is not set then a default restore client will be set to a backup client as usual. It is possible to define a dedicated restore job and run an automatic (scheduled) restore tests of your backups which will be redirected to the restore test Client.

always when the file to be restored already exists, it is deleted and then replaced by the copy that was backed up. This is the default value.

ifnewer if the backed up file (on tape) is newer than the existing file, the existing file is deleted and replaced by the back up.

ifolder if the backed up file (on tape) is older than the existing file, the existing file is deleted and replaced by the back up.

never if the backed up file already exists, Bacula skips restoring this file.

Prefix Links=<yes|no> If a `Where` path prefix is specified for a recovery job, apply it to absolute links as well. The default is no. When set to yes then while restoring files to an alternate directory, any absolute soft links will also be modified to point to the new alternate directory. Normally this is what is desired – i.e. everything is self consistent. However, if you wish to later move the files to their original locations, all files linked with absolute names will be broken.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs from the current Job resource that can run concurrently. Note, this directive limits only Jobs with the same name as the resource in which it appears. Any other restrictions on the maximum concurrent jobs such as in the Director, Client, or Storage resources will also apply in addition to the limit specified here. The default is set to 1, but you may set it to a larger number. We strongly recommend that you read the WARNING documented under Maximum Concurrent Jobs in the Director's resource.

Reschedule On Error = <yes|no> If this directive is enabled, and the job terminates in error, the job will be rescheduled as determined by the **Reschedule Interval** and **Reschedule Times** directives. If you cancel the job, it will not be rescheduled. The default is no (i.e. the job will not be rescheduled).

This specification can be useful for portables, laptops, or other machines that are not always connected to the network or switched on.

Reschedule Incomplete Jobs = <yes|no> If this directive is enabled, and the job terminates in incomplete status, the job will be rescheduled as determined by the **Reschedule Interval** and **Reschedule Times** directives. If you cancel the job, it will not be rescheduled. The default is yes (i.e. Incomplete jobs will be rescheduled).

Reschedule Interval = <time-specification> If you have specified **Reschedule On Error**=yes and the job terminates in error, it will be rescheduled after the interval of time specified by <time-specification>. See the time specification formats in the Configure chapter for details of time specifications. If no interval is specified, the job will not be rescheduled on error. The default Reschedule Interval is 30 minutes (1800 seconds).

Reschedule Times = <count> This directive specifies the maximum number of times to reschedule the job. If it is set to zero (0, the default) the job will be rescheduled an indefinite number of times.

Allow Incomplete Jobs = <yes|no> If this directive is disabled, and the job terminates in incomplete status, the data of the job will be discarded and the job will be marked in error. Bacula will treat this job like a regular job in error. The default is yes.

Allow Duplicate Jobs = <yes|no> A duplicate job in the sense we use it here means a second or subsequent job with the same name starts. This happens most frequently when the first job runs longer than expected because no tapes are available. The default is yes.

If this directive is enabled duplicate jobs will be run. If the directive is set to no then only one job of a given name may run at one time, and the action that Bacula takes to ensure only one job runs is determined by the other directives (see below).

If **Allow Duplicate Jobs** is set to no and two jobs are present and none of the three directives given below permit cancelling a job, then the current job (the second one started) will be cancelled.

Allow Higher Duplicates = <yes|no> This directive was implemented in version 5.0.0, but does not work as expected. If used, it should always be set to no. In later versions of Bacula the directive is disabled (disregarded).

Cancel Lower Level Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to no and this directive is set to yes, Bacula will choose between duplicated jobs the one with the highest level. For example, it will cancel a previous Incremental to run a Full backup. It works only for Backup jobs. The default is no. If the levels of the duplicated jobs are the same, nothing is done and the other Cancel XXX Duplicate directives will be examined.

Cancel Queued Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to no and if this directive is set to yes any job that is already queued to run but not yet running will be canceled. The default is no.

Cancel Running Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to no and if this directive is set to yes any job that is already running will be canceled. The default is no.

Run = <job-name> The **Run** directive (not to be confused with the Run option in a Schedule) allows you to start other jobs or to clone jobs. By using the cloning keywords (see below), you can backup the same data (or almost the same data) to two or more drives at the same time. The <job-name> is normally the same name as the current Job resource (thus creating a clone). However, it may be any Job name, so one job may start other related jobs.

The part after the equal sign must be enclosed in double quotes, and can contain any string or set of options (overrides) that you can specify when entering the `run` command from the console. For example **storage=DDS-4 ...**. In addition, there are two special

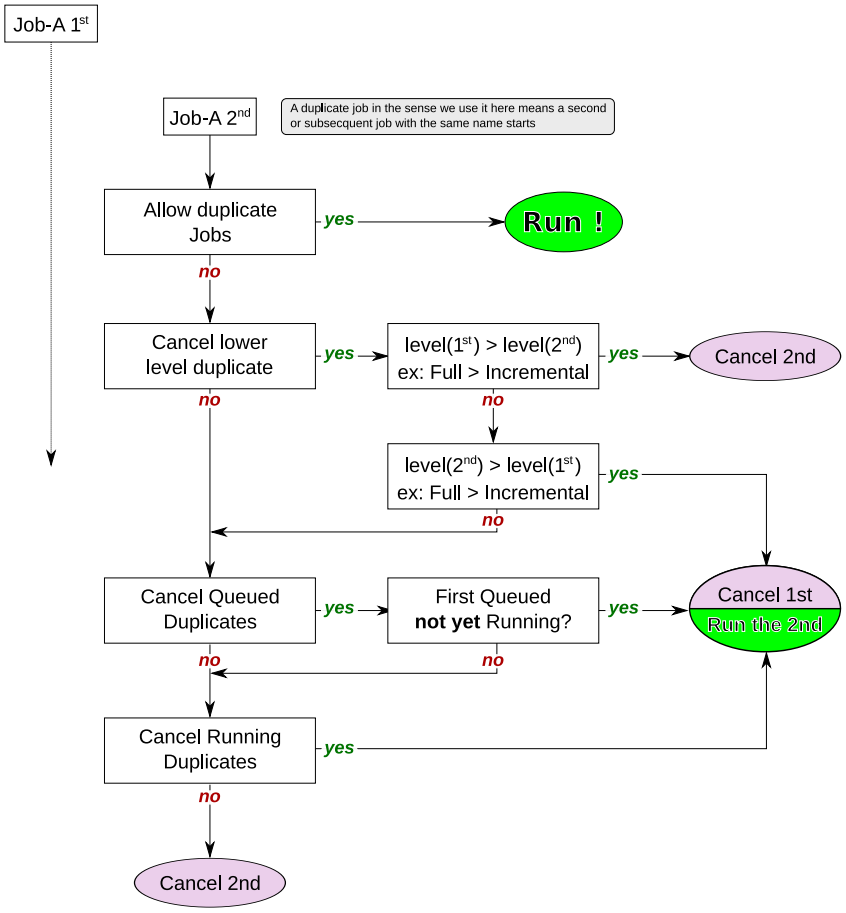


Figure 22.2: Allow Duplicate Jobs usage

keywords that permit you to clone the current job. They are **level=%l** and **since=%s**. The %l in the level keyword permits entering the actual level of the current job and the %s in the since keyword permits putting the same time for comparison as used on the current job. Note, in the case of the since keyword, the %s must be enclosed in double quotes, and thus they must be preceded by a backslash since they are already inside quotes. For example:

```
run = "Nightly-backup level=%l since=\"%s\" storage=DDS-4"
```

A cloned job will not start additional clones, so it is not possible to recurse.

Please note that all cloned jobs, as specified in the Run directives are submitted for running before the original job is run (while it is being initialized). This means that any clone job will actually start before the original job, and may even block the original job from starting until the original job finishes unless you allow multiple simultaneous jobs. Even if you set a lower priority on the clone job, if no other jobs are running, it will start before the original job.

If you are trying to prioritize jobs by using the clone feature (Run directive), you will find it much easier to do using a RunScript resource, or a RunBeforeJob directive.

Priority = <number> This directive permits you to control the order in which your jobs will be run by specifying a positive non-zero number. The higher the number, the lower the job priority. Assuming you are not running concurrent jobs, all queued jobs of priority 1 will run before queued jobs of priority 2 and so on, regardless of the original scheduling order.

The priority only affects waiting jobs that are queued to run, not jobs that are already running. If one or more jobs of priority 2 are already running, and a new job is scheduled with priority 1, the currently running priority 2 jobs must complete before the priority 1 job is run, unless **Allow Mixed Priority** is set.

The default priority is 10.

If you want to run concurrent jobs you should keep these points in mind:

- See **Running Concurrent Jobs** section (section 2.17 page 23) on how to setup concurrent jobs in the Bacula Community Edition Problems Resolution Guide.
- Bacula concurrently runs jobs of only one priority at a time. It will not simultaneously run a priority 1 and a priority 2 job.
- If Bacula is running a priority 2 job and a new priority 1 job is scheduled, it will wait until the running priority 2 job terminates even if the **Maximum Concurrent Jobs** settings would otherwise allow two jobs to run simultaneously.
- Suppose that bacula is running a priority 2 job and a new priority 1 job is scheduled and queued waiting for the running priority 2 job to terminate. If you then start a second priority 2 job, the waiting priority 1 job will prevent the new priority 2 job from running concurrently with the running priority 2 job. That is: as long as there is a higher priority job waiting to run, no new lower priority jobs will start even if the Maximum Concurrent Jobs settings would normally allow them to run. This ensures that higher priority jobs will be run as soon as possible.

If you have several jobs of different priority, it may not best to start them at exactly the same time, because Bacula must examine them one at a time. If by Bacula starts a lower priority job first, then it will run before your high priority jobs. If you experience this problem, you may avoid it by starting any higher priority jobs a few seconds before lower priority ones. This insures that Bacula will examine the jobs in the correct order, and that your priority scheme will be respected.

Allow Mixed Priority = <yes|no> This directive is only implemented in version 2.5 and later. When set to yes (default no), this job may run even if lower priority jobs are already running. This means a high priority job will not have to wait for other jobs to finish before starting. The scheduler will only mix priorities when all running jobs have this set to true.

Note that only higher priority jobs will start early. Suppose the director will allow two concurrent jobs, and that two jobs with priority 10 are running, with two more in the

queue. If a job with priority 5 is added to the queue, it will be run as soon as one of the running jobs finishes. However, new priority 10 jobs will not be run until the priority 5 job has finished.

The following is an example of a valid Job resource definition:

```
Job {
  Name = "Minou"
  Type = Backup
  Level = Incremental                 # default
  Client = Minou
  FileSet="Minou Full Set"
  Storage = DLTDrive
  Pool = Default
  Schedule = "MinouWeeklyCycle"
  Messages = Standard
}
```

22.4 The JobDefs Resource

The JobDefs resource permits all the same directives that can appear in a Job resource. However, a JobDefs resource does not create a Job, rather it can be referenced within a Job to provide defaults for that Job. This permits you to concisely define several nearly identical Jobs, each one referencing a JobDefs resource which contains the defaults. Only the changes from the defaults need to be mentioned in each Job.

22.5 The Schedule Resource

The Schedule resource provides a means of automatically scheduling a Job as well as the ability to override the default Level, Pool, Storage and Messages resources. If a Schedule resource is not referenced in a Job, the Job can only be run manually. In general, you specify an action to be taken and when.

Schedule Start of the Schedule directives. No `Schedule` resource is required, but you will need at least one if you want Jobs to be automatically started.

Name = <name> The name of the schedule being defined. The Name directive is required.

Enabled = <yes|no> This directive allows you to enable or disable the `Schedule` resource.

Run = <Job-overrides> <Date-time-specification> The Run directive defines when a Job is to be run, and what overrides if any to apply. You may specify multiple **Run** directives within a `Schedule` resource. If you do, they will all be applied (i.e. multiple schedules). If you have two **Run** directives that start at the same time, two Jobs will start at the same time (well, within one second of each other).

The **Job-overrides** permit overriding the Level, the Storage, the Messages, and the Pool specifications provided in the Job resource. In addition, the **FullPool**, the **IncrementalPool**, and the **DifferentialPool** specifications permit overriding the Pool specification according to what backup Job Level is in effect.

By the use of overrides, you may customize a particular Job. For example, you may specify a Messages override for your Incremental backups that outputs messages to a log file, but for your weekly or monthly Full backups, you may send the output by email by using a different Messages override.

Job-overrides are specified as: **keyword=value** where the keyword is Level, Storage, Messages, Pool, FullPool, DifferentialPool, or IncrementalPool, and the value is as defined

on the respective directive formats for the Job resource. You may specify multiple **Job-overrides** on one **Run** directive by separating them with one or more spaces or by separating them with a trailing comma. For example:

Level=Full is all files in the FileSet whether or not they have changed.

Level=Incremental is all files that have changed since the last backup.

Pool=Weekly specifies to use the Pool named **Weekly**.

Storage=DLT_Drive specifies to use **DLT_Drive** for the storage device.

Messages=Verbose specifies to use the **Verbose** message resource for the Job.

FullPool=Full specifies to use the Pool named **Full** if the job is a full backup, or is upgraded from another type to a Full backup.

DifferentialPool=Differential specifies to use the Pool named **Differential** if the job is a differential backup.

IncrementalPool=Incremental specifies to use the Pool named **Incremental** if the job is an incremental backup.

Next Pool = <pool-specification> The **Next Pool** directive specifies the pool to which Jobs will be migrated. This directive is used to define the Pool into which the data will be migrated.

Priority = <number> This directive permits you to control the order in which your jobs will be run by specifying a positive non-zero number. The higher the number, the lower the job priority. Assuming you are not running concurrent jobs, all queued jobs of priority 1 will run before queued jobs of priority 2 and so on, regardless of the original scheduling order.

The priority only affects waiting jobs that are queued to run, not jobs that are already running. If one or more jobs of priority 2 are already running, and a new job is scheduled with priority 1, the currently running priority 2 jobs must complete before the priority 1 job is run, unless **Allow Mixed Priority** is set.

The default priority is 10.

See 22.3 on page 240 for more information.

<Date-time-specification> determines when the Job is to be run. The specification is a repetition, and as a default Bacula is set to run a job at the beginning of the hour of every hour of every day of every week of every month of every year. This is not normally what you want, so you must specify or limit when you want the job to run. Any specification given is assumed to be repetitive in nature and will serve to override or limit the default repetition. This is done by specifying masks or times for the hour, day of the month, day of the week, week of the month, week of the year, and month when you want the job to run. By specifying one or more of the above, you can define a schedule to repeat at almost any frequency you want.

Basically, you must supply a **month**, **day**, **hour**, and **minute** the Job is to be run. Of these four items to be specified, **day** is special in that you may either specify a day of the month such as 1, 2, ... 31, or you may specify a day of the week such as Monday, Tuesday, ... Sunday. Finally, you may also specify a week qualifier to restrict the schedule to the first, second, third, fourth, fifth or sixth week of the month.

For example, if you specify only a day of the week, such as Tuesday the Job will be run every hour of every Tuesday of every Month. That is the month and hour remain set to the defaults of every month and all hours.

Note, by default with no other specification, your job will run at the beginning of every hour. If you wish your job to run more than once in any given hour, you will need to specify multiple **Run** specifications each with a different minute.

The date/time to run the Job can be specified in the following way in pseudo-BNF:

```
<void-keyword>    = on
<at-keyword>      = at
<week-keyword>    = 1st | 2nd | 3rd | 4th | 5th | 6th | first |
                    second | third | fourth | fifth
<wday-keyword>    = sun | mon | tue | wed | thu | fri | sat |
```

```
                    sunday | monday | tuesday | wednesday |
                    thursday | friday | saturday
<week-of-year-keyword> = w00 | w01 | ... w52 | w53
<month-keyword>   = jan | feb | mar | apr | may | jun | jul |
                    aug | sep | oct | nov | dec | january |
                    february | ... | december
<daily-keyword>   = daily
<weekly-keyword>  = weekly
<monthly-keyword> = monthly
<hourly-keyword>  = hourly
<digit>           = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<number>          = <digit> | <digit><number>
<12hour>          = 0 | 1 | 2 | ... 12
<hour>            = 0 | 1 | 2 | ... 23
<minute>          = 0 | 1 | 2 | ... 59
<day>             = 1 | 2 | ... 31 | lastday
<time>            = <hour>:<minute> |
                    <12hour>:<minute>am |
                    <12hour>:<minute>pm
<time-spec>       = <at-keyword> <time> |
                    <hourly-keyword>
<date-keyword>    = <void-keyword>  <weekly-keyword>
<day-range>       = <day>-<day>
<month-range>     = <month-keyword>-<month-keyword>
<wday-range>      = <wday-keyword>-<wday-keyword>
<range>           = <day-range> | <month-range> |
                          <wday-range>
<date>            = <date-keyword> | <day> | <range>
<date-spec>       = <date> | <date-spec>
<day-spec>        = <day> | <wday-keyword> |
                    <day> | <wday-range> |
                    <week-keyword> <wday-keyword> |
                    <week-keyword> <wday-range> |
                    <daily-keyword>
<month-spec>      = <month-keyword> | <month-range> |
                    <monthly-keyword>
<date-time-spec>  = <month-spec> <day-spec> <time-spec>
```

Note, the Week of Year specification wnn follows the ISO standard definition of the week of the year, where Week 1 is the week in which the first Thursday of the year occurs, or alternatively, the week which contains the 4th of January. Weeks are numbered w01 to w53. w00 for Bacula is the week that precedes the first ISO week (i.e. has the first few days of the year if any occur before Thursday). w00 is not defined by the ISO specification. A week starts with Monday and ends with Sunday.

According to the US National Institute of Standards and Technology (NIST), 12am and 12pm are ambiguous and can be defined to anything. However, 12:01am is the same as 00:01 and 12:01pm is the same as 12:01, so Bacula defines 12am as 00:00 (midnight) and 12pm as 12:00 (noon). You can avoid this abiguity (confusion) by using 24 hour time specifications (i.e. no am/pm). This is the definition in Bacula version 2.0.3 and later.

An example schedule resource that is named WeeklyCycle and runs a job with level full each Sunday at 2:05am and an incremental job Monday through Saturday at 2:05am is:

```
Schedule {
  Name = "WeeklyCycle"
  Run = Level=Full sun at 2:05
  Run = Level=Incremental mon-sat at 2:05
}
```

An example of a possible monthly cycle is as follows:

```
Schedule {
  Name = "MonthlyCycle"
  Run = Level=Full Pool=Monthly 1st sun at 2:05
  Run = Level=Differential 2nd-5th sun at 2:05
  Run = Level=Incremental Pool=Daily mon-sat at 2:05
}
```

The first of every month:

```
Schedule {
  Name = "First"
  Run = Level=Full on 1 at 2:05
  Run = Level=Incremental on 2-31 at 2:05
}
```

The last day of February (28th or 29th), on May 31st and September 30rd at 20:00. In `show schedule` output, `LastDay is printed as "31"`

```
Schedule {
  Name = "Last"
  Run = Level=Full on lastday  Feb, May, Sep at 20:00
}
```

Every 10 minutes:

```
Schedule {
  Name = "TenMinutes"
  Run = Level=Full hourly at 0:05
  Run = Level=Full hourly at 0:15
  Run = Level=Full hourly at 0:25
  Run = Level=Full hourly at 0:35
  Run = Level=Full hourly at 0:45
  Run = Level=Full hourly at 0:55
}
```

22.6 Technical Notes on Schedules

Internally Bacula keeps a schedule as a bit mask. There are six masks and a minute field to each schedule. The masks are hour, day of the month (mday), month, day of the week (wday), week of the month (wom), and week of the year (woy). The schedule is initialized to have the bits of each of these masks set, which means that at the beginning of every hour, the job will run. When you specify a month for the first time, the mask will be cleared and the bit corresponding to your selected month will be selected. If you specify a second month, the bit corresponding to it will also be added to the mask. Thus when Bacula checks the masks to see if the bits are set corresponding to the current time, your job will run only in the two months you have set. Likewise, if you set a time (hour), the hour mask will be cleared, and the hour you specify will be set in the bit mask and the minutes will be stored in the minute field.

For any schedule you have defined, you can see how these bits are set by doing a `show schedules` command in the Console program. Please note that the bit mask is zero based, and Sunday is the first day of the week (bit zero).

22.7 The FileSet Resource

The FileSet resource defines what files are to be included or excluded in a backup job. A **FileSet** resource is required for each backup Job. It consists of a list of files or directories to be included, a list of files or directories to be excluded and the various backup options such as compression, encryption, and signatures that are to be applied to each file.

Any change to the list of the included files will cause Bacula to automatically create a new FileSet (defined by the name and an MD5 checksum of the Include/Exclude contents). Each time a new FileSet is created, Bacula will ensure that the next backup is always a Full save.

Bacula is designed to handle most character sets of the world, US ASCII, German, French, Chinese, ... However, it does this by encoding everything in UTF-8, and it expects all configuration files (including those read on Win32 machines) to be in UTF-8 format. UTF-8 is typically the default on Linux machines, but not on all Unix machines, nor on Windows, so you must take some care to ensure that your locale is set properly before starting Bacula. On most modern Win32 machines, you can edit the conf files with **notepad** and choose output encoding UTF-8.

To ensure that Bacula configuration files can be correctly read including foreign characters the **LANG** environment variable must end in **.UTF-8**. A full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary.

Bacula assumes that all filenames are in UTF-8 format on Linux and Unix machines. On Win32 they are in Unicode (UTF-16), and will be automatically converted to UTF-8 format.

FileSet Start of the FileSet resource. One **FileSet** resource must be defined for each Backup job.

Name = <name> The name of the FileSet resource. This directive is required.

Ignore FileSet Changes = <yes|no> Normally, if you modify the FileSet Include or Exclude lists, the next backup will be forced to a Full so that Bacula can guarantee that any additions or deletions are properly saved.

We strongly recommend against setting this directive to yes, since doing so may cause you to have an incomplete set of backups.

If this directive is set to yes, any changes you make to the FileSet Include or Exclude lists, will not force a Full during subsequent backups. Note that any changes to Options resources in the FileSet are not considered by this directive. You can use the Accurate mode for this to be treated correctly, or schedule a new Full backup manually.

The default is no, in which case, if you change the Include or Exclude lists, Bacula will force a Full backup to ensure that everything is properly backed up.

Enable VSS = <yes|no> If this directive is set to yes the File daemon will be notified that the user wants to use a Volume Snapshot Service (VSS) backup for this job. The default is yes. This directive is effective only for VSS enabled Win32 File daemons. It permits a consistent copy of open files to be made for cooperating writer applications, and for applications that are not VSS aware, Bacula can at least copy open files. The Volume Snapshot Servicewill only be done on Windows drives where the drive (e.g. C:, D:, ...) is explicitly mentioned in a **File** directive. For more information, please see the Windows chapter of this manual.

Enable Snapshot = <yes|no> If this directive is set to **yes** the File daemon will be notified that the user wants to use the Snapshot Engine for this job. The default is no. This directive is effective only for Snapshot enabled Unix File daemons. It permits a consistent copy of open files to be made for cooperating applications. The `bsnapshot` tool should be installed on the Client.

Include {Options {<file-options>} ... ; <file-list> }

Exclude {<file-list> }

The Include resource must contain a list of directories and/or files to be processed in the backup job. Normally, all files found in all subdirectories of any directory in the Include File list will be backed up. Note, see below for the definition of <file-list>. The Include resource may also contain one or more Options resources that specify options such as compression to be applied to all or any subset of the files found when processing the file-list for backup. Please see below for more details concerning Options resources.

There can be any number of **Include** resources within the FileSet, each having its own list of directories or files to be backed up and the backup options defined by one or more Options

resources. The **file-list** consists of one file or directory name per line. Directory names should be specified without a trailing slash with Unix path notation.

Windows users, please take note to specify directories (even c:/...) in Unix path notation. If you use Windows conventions, you will most likely not be able to restore your files due to the fact that the Windows path separator was defined as an escape character long before Windows existed, and Bacula adheres to that convention (i.e.
means the next character appears as itself).

You should always specify a full path for every directory and file that you list in the FileSet. In addition, on Windows machines, you should **always** prefix the directory or filename with the drive specification (e.g. c:/xxx) using Unix directory name separators (forward slash). The drive letter itself can be upper or lower case (e.g. c:/xxx or C:/xxx).

Bacula's default for processing directories is to recursively descend in the directory saving all files and subdirectories. Bacula will not by default cross filesystems (or mount points in Unix parlance). This means that if you specify the root partition (e.g. /), Bacula will save only the root partition and not any of the other mounted filesystems. Similarly on Windows systems, you must explicitly specify each of the drives you want saved (e.g. c:/ and d:/ ...). In addition, at least for Windows systems, you will most likely want to enclose each specification within double quotes particularly if the directory (or file) name contains spaces. The df command on Unix systems will show you which mount points you must specify to save everything. See below for an example.

Take special care not to include a directory twice or Bacula will backup the same files two times wasting a lot of space on your archive device. Including a directory twice is very easy to do. For example:

```
Include {
  Options {compression=GZIP }
  File = /
  File = /usr
}
```

on a Unix system where /usr is a subdirectory (rather than a mounted filesystem) will cause /usr to be backed up twice.

Please take note of the following items in the FileSet syntax:

1. There is no equal sign (=) after the Include and before the opening brace ({). The same is true for the Exclude.
2. Each directory (or filename) to be included or excluded is preceded by a **File =**. Previously they were simply listed on separate lines.
3. The options that previously appeared on the Include line now must be specified within their own Options resource.
4. The Exclude resource does not accept Options.
5. When using wild-cards or regular expressions, directory names are always terminated with a slash (/) and filenames have no trailing slash.

The Options resource is optional, but when specified, it will contain a list of **keyword=value** options to be applied to the file-list. See below for the definition of file-list. Multiple Options resources may be specified one after another. As the files are found in the specified directories, the Options will applied to the filenames to determine if and how the file should be backed up. The wildcard and regular expression pattern matching parts of the Options resources are checked in the order they are specified in the FileSet until the first one that matches. Once one matches, the compression and other flags within the Options specification will apply to the pattern matched.

A key point is that in the absence of an Option or no other Option is matched, every file is accepted for backing up. This means that if you want to exclude something, you must explicitly specify an Option with an **exclude = yes** and some pattern matching.

Once Bacula determines that the Options resource matches the file under consideration, that file will be saved without looking at any other Options resources that may be present. This means that any wild cards must appear before an Options resource without wild cards.

If for some reason, Bacula checks all the Options resources to a file under consideration for backup, but there are no matches (generally because of wild cards that don't match), Bacula as a default will then backup the file. This is quite logical if you consider the case of no Options clause is specified, where you want everything to be backed up, and it is important to keep in mind when excluding as mentioned above.

However, one additional point is that in the case that no match was found, Bacula will use the options found in the last Options resource. As a consequence, if you want a particular set of "default" options, you should put them in an Options resource after any other Options.

It is a good idea to put all your wild-card and regex expressions inside double quotes to prevent conf file scanning problems.

This is perhaps a bit overwhelming, so there are a number of examples included below to illustrate how this works.

You find yourself using a lot of Regex statements, which will cost quite a lot of CPU time, we recommend you simplify them if you can, or better yet convert them to Wild statements which are much more efficient.

The directives within an Options resource may be one of the following:

compression=GZIP All files saved will be software compressed using the GNU ZIP compression format. The compression is done on a file by file basis by the File daemon. If there is a problem reading the tape in a single record of a file, it will at most affect that file and none of the other files on the tape. Normally this option is **not** needed if you have a modern tape drive as the drive will do its own compression. In fact, if you specify software compression at the same time you have hardware compression turned on, your files may actually take more space on the volume.

Software compression is very important if you are writing your Volumes to a file, and it can also be helpful if you have a fast computer but a slow network, otherwise it is generally better to rely your tape drive's hardware compression. As noted above, it is not generally a good idea to do both software and hardware compression.

Specifying **GZIP** uses the default compression level 6 (i.e. **GZIP** is identical to **GZIP6**). If you want a different compression level (1 through 9), you can specify it by appending the level number with no intervening spaces to **GZIP**. Thus **compression=GZIP1** would give minimum compression but the fastest algorithm, and **compression=GZIP9** would give the highest level of compression, but requires more computation. According to the GZIP documentation, compression levels greater than six generally give very little extra compression and are rather CPU intensive.

You can overwrite this option per Storage resource with AllowCompression option.

compression=LZO All files saved will be software compressed using the LZO compression format. The compression is done on a file by file basis by the File daemon. Everything else about GZIP is true for LZO.

LZO provides much faster compression and decompression speed but lower compression ratio than GZIP. If your CPU is fast enough you should be able to compress your data without making the backup duration longer.

Note that bacula only use one compression level LZO1X-1 specified by LZO.

You can overwrite this option per Storage resource with AllowCompression option.

signature=SHA1 An SHA1 signature will be computed for all The SHA1 algorithm is purported to be some what slower than the MD5 algorithm, but at the same time is significantly better from a cryptographic point of view (i.e. much fewer collisions, much lower probability of being hacked.) It adds four more bytes than the MD5 signature. We strongly recommend that either this option or MD5 be specified as a default for all files. Note, only one of the two options MD5 or SHA1 can be computed for any file.

signature=SHA256 A SHA256 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the SHA256 signature adds 44 more bytes per file to your catalog.

signature=SHA512 A SHA256 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the SHA512 signature adds 87 more bytes per file to your catalog.

signature=MD5 An MD5 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the MD5 signature adds 16 more bytes per file to your catalog. We strongly recommend that this option or the SHA1 option be specified as a default for all files.

dedup=none|storage|bothsides The **Dedup** Fileset option can use the following values:

* **storage** All the deduplication work is done on Storage Daemon side if the device type is dedup. (Default value)

none Force File Daemon and Storage Daemon to not use deduplication even if the device type is **dedup**.

bothsides The deduplication work is done on the FD and the SD if the device type is **dedup**.

basejob=<options> The options letters specified are used when running a **Backup Level=Full** with BaseJobs. The options letters are the same as in the **accurate=** option below.

accurate=<options> The options letters specified are used when running a **Backup Level=Incremental/Differential** in Accurate mode. The options letters are the same as in the **verify=** option below with the addition of the following options:

A Always backup files

M Look mtime/ctime like normal incremental backup

o Save only the metadata of a file when possible[3].

The 'o' option should be used in conjunction with one of the signature checking options (1, 2, 3, or 5). When the option is specified, the signature is computed only for files that have one of the other accurate options specified triggering a backup of the file (for example an inode change, a permission change, etc...). In cases where only the file's metadata has changed (ie: the signature is identical), only the file's attributes will be backed up. If the file's data has been changed (hence a different singature), the file will be backed up in the usual way

verify=<options> The options letters specified are used when running a **Verify Level=Catalog** as well as the **DiskToCatalog** level job. The options letters may be any combination of the following:

i compare the inodes

p compare the permission bits

n compare the number of links

u compare the user id

g compare the group id

s compare the size

a compare the access time

[3]Available from version 13.0.0

m compare the modification time (st_mtime)

c compare the change time (st_ctime)

d report file size decreases

5 compare the MD5 signature

1 compare the SHA1 signature

2 compare the SHA256 signature

3 compare the SHA512 signature

A useful set of general options on the **Level=Catalog** or **Level=DiskToCatalog** verify is **pins5** i.e. compare permission bits, then inodes, number of links, size, and finally MD5 changes.

To save some space when backing up files which have only metadata part changed (e.g. changed permissions) the 'pino5' options could be used (or some other combination with the 'o' flag). It will then compare permision bits, inotes, number of links and if any of it changes it will also compute file's singature to verify if only metadata need to be backed up or is it needed to update file's contents.

onefs=<yes|no> If set to yes (the default), **Bacula** will remain on a single file system. That is it will not backup file systems that are mounted on a subdirectory. If you are using a *nix system, you may not even be aware that there are several different filesystems as they are often automatically mounted by the OS (e.g. /dev, /net, /sys, /proc, ...). With Bacula 1.38.0 or later, it will inform you when it decides not to traverse into another filesystem. This can be very useful if you forgot to backup a particular partition. An example of the informational message in the job report is:

```
rufus-fd: /misc is a different filesystem. Will not descend from / into /misc
rufus-fd: /net is a different filesystem. Will not descend from / into /net
rufus-fd: /var/lib/nfs/rpc_pipefs is a different filesystem. Will not descend from /var/lib/nfs into /va
rufus-fd: /selinux is a different filesystem. Will not descend from / into /selinux
rufus-fd: /sys is a different filesystem. Will not descend from / into /sys
rufus-fd: /dev is a different filesystem. Will not descend from / into /dev
rufus-fd: /home is a different filesystem. Will not descend from / into /home
```

Note: in older versions of Bacula, the above message was of the form:

```
Filesystem change prohibited. Will not descend into /misc
```

If you wish to backup multiple filesystems, you can explicitly list each filesystem you want saved. Otherwise, if you set the onefs option to **no**, Bacula will backup all mounted file systems (i.e. traverse mount points) that are found within the **FileSet**. Thus if you have NFS or Samba file systems mounted on a directory listed in your FileSet, they will also be backed up. Normally, it is preferable to set **onefs=yes** and to explicitly name each filesystem you want backed up. Explicitly naming the filesystems you want backed up avoids the possibility of getting into a infinite loop recursing filesystems. Another possibility is to use **onefs=no** and to set **fstype=ext2, ...**. See the example below for more details.

If you think that Bacula should be backing up a particular directory and it is not, and you have **onefs=no** set, before you complain, please do:

```
stat /
stat <filesystem>
```

where you replace **filesystem** with the one in question. If the **Device:** number is different for / and for your filesystem, then they are on different filesystems. E.g.

```
stat /
  File: '/'
  Size: 4096            Blocks: 16         IO Block: 4096   directory
Device: 302h/770d       Inode: 2           Links: 26
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2005-11-10 12:28:01.000000000 +0100
Modify: 2005-09-27 17:52:32.000000000 +0200
Change: 2005-09-27 17:52:32.000000000 +0200
```

```
stat /net
  File: '/home'
  Size: 4096            Blocks: 16         IO Block: 4096   directory
Device: 308h/776d       Inode: 2           Links: 7
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2005-11-10 12:28:02.000000000 +0100
Modify: 2005-11-06 12:36:48.000000000 +0100
Change: 2005-11-06 12:36:48.000000000 +0100
```

Also be aware that even if you include /home in your list of files to backup, as you most likely should, you will get the informational message that "/home is a different filesystem" when Bacula is processing the / directory. This message does not indicate an error. This message means that while examining the **File =** referred to in the second part of the message, Bacula will not descend into the directory mentioned in the first part of the message. However, it is possible that the separate filesystem will be backed up despite the message. For example, consider the following FileSet:

```
File = /
File = /var
```

where /var is a separate filesystem. In this example, you will get a message saying that Bacula will not decend from / into /var. But it is important to realise that Bacula will descend into /var from the second File directive shown above. In effect, the warning is bogus, but it is supplied to alert you to possible omissions from your FileSet. In this example, /var will be backed up. If you changed the FileSet such that it did not specify /var, then /var will not be backed up.

honor nodump flag=<yes|no> If your file system supports the **nodump** flag (e. g. most BSD-derived systems) Bacula will honor the setting of the flag when this option is set to **yes**. Files having this flag set will not be included in the backup and will not show up in the catalog. For directories with the **nodump** flag set recursion is turned off and the directory will be listed in the catalog. If the **honor nodump flag** option is not defined or set to **no** every file and directory will be eligible for backup.

portable=<yes|no> If set to **yes** (default is no), the Bacula File daemon will backup Win32 files in a portable format, but not all Win32 file attributes will be saved and restored. By default, this option is set to no, which means that on Win32 systems, the data will be backed up using Windows BackupRead API calls and all the security and ownership attributes will be properly backed up (and restored). However this format is not portable to other systems – e.g. Unix, Win95/98/Me. When backing up Unix systems, this option is ignored, and unless you have a specific need to have portable backups, we recommend accept the default (**no**) so that the maximum information concerning your Windows files is saved.

recurse=<yes|no> If set to yes (the default), Bacula will recurse (or descend) into all subdirectories found unless the directory is explicitly excluded using an **exclude** definition. If you set **recurse=no**, Bacula will save the subdirectory entries, but not descend into the subdirectories, and thus will not save the files or directories contained in the subdirectories. Normally, you will want the default (yes).

sparse=<yes|no> Enable special code that checks for sparse files such as created by ndbm. The default is no, so no checks are made for sparse files. You may specify **sparse=yes** even on files that are not sparse file. No harm will be done, but there will be a small additional overhead to check for buffers of all zero, and if there is a 32K block of all zeros (see below), that block will become a hole in the file, which may not be desirable if the original file was not a sparse file.

Restrictions: Bacula reads files in 64K buffers. If the whole buffer is zero, it will be treated as a sparse block and not written to tape. However, if any part of the buffer is non-zero, the whole buffer will be written to tape, possibly including some disk sectors (generally 4098 bytes) that are all zero. As a consequence, Bacula's detection of sparse blocks is in 64K increments rather than the system block size. If anyone considers this to be a real problem, please send in a request for change with the reason.

If you are not familiar with sparse files, an example is say a file where you wrote 512 bytes at address zero, then 512 bytes at address 1 million. The operating system will allocate only two blocks, and the empty space or hole will have nothing allocated. However, when you read the sparse file and read the addresses where nothing was written, the OS will return all zeros as if the space were allocated, and if you backup such a file, a lot of space will be used to write zeros to the volume. Worse yet, when you restore the file, all the previously empty space will now be allocated using much more disk space. By turning on the **sparse** option, Bacula will specifically look for empty space in the file, and any empty space will not be written to the Volume, nor will it be restored. The price to pay for this is that Bacula must search each block it reads before writing it. On a slow system, this may be important. If you suspect you have sparse files, you should benchmark the difference or set sparse for only those files that are really sparse.

You probably should not use this option on files or raw disk devices that are not really sparse files (i.e. have holes in them).

readfifo=<yes|no> If enabled, tells the Client to read the data on a backup and write the data on a restore to any FIFO (pipe) that is explicitly mentioned in the FileSet. In this case, you must have a program already running that writes into the FIFO for a backup or reads from the FIFO on a restore. This can be accomplished with the **RunBeforeJob** directive. If this is not the case, Bacula will hang indefinitely on reading/writing the FIFO. When this is not enabled (no, the default), the Client simply saves the directory entry for the FIFO.

Unfortunately, when Bacula runs a RunBeforeJob, it waits until that script terminates, and if the script accesses the FIFO to write into the it, the Bacula job will block and everything will stall. However, Vladimir Stavrinov as supplied tip that allows this feature to work correctly. He simply adds the following to the beginning of the RunBeforeJob script:

```
exec > /dev/null
```

noatime=<yes|no> If enabled, and if your Operating System supports the O_NOATIME file open flag, Bacula will open all files to be backed up with this option. It makes it possible to read a file without updating the inode atime (and also without the inode ctime update which happens if you try to set the atime back to its previous value). It also prevents a race condition when two programs are reading the same file, but only one does not want to change the atime. It's most useful for backup programs and file integrity checkers (and bacula can fit on both categories).

This option is particularly useful for sites where users are sensitive to their MailBox file access time. It replaces both the **keepatime** option without the inconveniences of that option (see below).

If your Operating System does not support this option, it will be silently ignored by Bacula.

mtimeonly=<yes|no> If enabled, tells the Client that the selection of files during Incremental and Differential backups should based only on the st_mtime value in the stat() packet. The default is no which means that the selection of files to be backed up will be based on both the st_mtime and the st_ctime values. In general, it is not recommended to use this option.

keepatime=<yes|no> The default is no. When enabled, Bacula will reset the st_atime (access time) field of files that it backs up to their value prior to the backup. This option is not generally recommended as there are very few programs that use st_atime, and the backup overhead is increased because of the additional system call necessary to reset the times. However, for some files, such as mailboxes, when Bacula backs up the file, the user will notice that someone (Bacula) has accessed the file. In this, case keepatime can be useful. (I'm not sure this works on Win32).

Note, if you use this feature, when Bacula resets the access time, the change time (st_ctime) will automatically be modified by the system, so on the next incremental job, the file will be backed up even if it has not changed. As a consequence, you will probably also want to use **mtimeonly = yes** as well as keepatime (thanks to Rudolf Cejka for this tip).

checkfilechanges=<yes|no> If enabled, the Client will check size, age of each file after their backup to see if they have changed during backup. If time or size mismatch, an error will raise. The default value is yes.

```
zog-fd: Client1.2007-03-31_09.46.21 Error: /tmp/test mtime changed during backup.
```

In general, it is recommended to use this option to backup regular files. It is not compatible with fifo or plugin streams.

hardlinks=<yes|no> When enabled (yes, the default), this directive will cause hard links to be backed up. However, the File daemon keeps track of hard linked files and will backup the data only once. The process of keeping track of the hard links can be quite expensive if you have lots of them (tens of thousands or more). This doesn't occur on normal Unix systems, but if you use a program like BackupPC, it can create hundreds of thousands, or even millions of hard links. Backups become very long and the File daemon will consume a lot of CPU power checking hard links. In such a case, set **hardlinks=no** and hard links will not be backed up. Note, using this option will most likely backup more data and on a restore the file system will not be restored identically to the original.

wild=<string> Specifies a wild-card string to be applied to the filenames and directory names. Note, if **Exclude** is not enabled, the wild-card will select which files are to be included. If **Exclude=yes** is specified, the wild-card will select which files are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the **Utilities** chapter (chapter 1.14 page 18) of the Bacula Community Edition Utility programs for more information. You can also test your full FileSet definition by using the `estimate` command (command 1.5 page 7) in the Bacula Community Edition Console manual. It is recommended to enclose the string in double quotes.

enhancedwild=<yes|no> The `Enhanced Wild` directive controls how path name matching with wildcards is done.

The default is `no`, which makes wildcards not match match path separators. If set to `yes`, an asterisk, question mark, or a bracketed slash will also be matched by a slash. In other words, wildcards will then span path hierarchy.

wilddir=<string> Specifies a wild-card string to be applied to directory names only. No filenames will be matched by this directive. Note, if **Exclude** is not enabled, the wild-card will select directories to be included. If **Exclude=yes** is specified, the wild-card will select which directories are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the **Utilities** chapter (chapter 1.14 page 18) of the Bacula Community Edition Utility programs for more information. You can also test your full FileSet definition by using the `estimate` command (command 1.5 page 7) in the Bacula Community Edition Console manual. An example of excluding with the WildDir option on Win32 machines is presented below.

wildfile=<string> Specifies a wild-card string to be applied to non-directories. That is no directory entries will be matched by this directive. However, note that the match is done against the full path and filename, so your wild-card string must take into account that filenames are preceded by the full path. If **Exclude** is not enabled, the wild-card will select which files are to be included. If **Exclude=yes** is specified, the wild-card will select which files are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches.

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the **Utilities** chapter (chapter 1.14 page 18) of the Bacula Community Edition Utility programs for more information. You can also test your full FileSet definition by using the `estimate` command (command 1.5 page 7) in the Bacula Community Edition Console manual. An example of excluding with the WildFile option on Win32 machines is presented below.

regex=<string> Specifies a POSIX extended regular expression to be applied to the filenames and directory names, which include the full path. If **Exclude** is not enabled, the regex will select which files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified within an Options resource, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the **Utilities** chapter (chapter 1.14 page 18) of the Bacula Community Edition Utility programs for more information. You can also test your full FileSet definition by using the `estimate` command (command 1.5 page 7) in the Bacula Community Edition Console manual.

You find yourself using a lot of Regex statements, which will cost quite a lot of CPU time, we recommend you simplify them if you can, or better yet convert them to Wild statements which are much more efficient.

regexfile=<string> Specifies a POSIX extended regular expression to be applied to non-directories. No directories will be matched by this directive. However, note that the match is done against the full path and filename, so your regex string must take into account that filenames are preceded by the full path. If **Exclude** is not enabled, the regex will select which files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified, and they will be applied in turn until the first one that matches.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the **bregex** command (command 1.13 page 18) of the Bacula Community Edition Utility programs more.

regexdir=<string> Specifies a POSIX extended regular expression to be applied to directory names only. No filenames will be matched by this directive. Note, if **Exclude** is not enabled, the regex will select directories files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the **bregex** command (command 1.13 page 18) of the Bacula Community Edition Utility programs more.

exclude=<yes|no> The default is no. When enabled, any files matched within the Options will be excluded from the backup.

aclsupport=<yes|no> The default is no. If this option is set to yes, and you have the POSIX **libacl** installed on your Linux system, Bacula will backup the file and directory Unix Access Control Lists (ACLs) as defined in IEEE Std 1003.1e draft 17 and "POSIX.1e" (abandoned). This feature is available on Unix systems only and requires the Linux ACL library. Bacula is automatically compiled with ACL support if the **libacl** library is installed on your Linux system (shown in config.out). While restoring the files Bacula will try to restore the ACLs, if there is no ACL support available on the system, Bacula restores the files and directories

but not the ACL information. Please note, if you backup an EXT3 or XFS filesystem with ACLs, then you restore them to a different filesystem (perhaps reiserfs) that does not have ACLs, the ACLs will be ignored.

For other operating systems there is support for either POSIX ACLs or the more extensible NFSv4 ACLs.

The ACL stream format between Operation Systems is **not** compatible so for example an ACL saved on Linux cannot be restored on Solaris.

The following Operating Systems are currently supported:

1. AIX (pre-5.3 (POSIX) and post 5.3 (POSIX and NFSv4) ACLs)
2. Darwin
3. FreeBSD (POSIX and NFSv4/ZFS ACLs)
4. HPUX
5. IRIX
6. Linux (POSIX and GPFS)
7. Solaris (POSIX and NFSv4/ZFS ACLs)
8. Tru64

xattrsupport=<yes|no> The default is no. If this option is set to yes, and your operating system support either so called Extended Attributes or Extensible Attributes Bacula will backup the file and directory XATTR data. This feature is available on UNIX only and depends on support of some specific library calls in libc.

The XATTR stream format between Operating Systems is **not** compatible so an XATTR saved on Linux cannot for example be restored on Solaris.

On some operating systems ACLs are also stored as Extended Attributes (Linux, Darwin, FreeBSD) Bacula checks if you have the aclsupport option enabled and if so will not save the same info when saving extended attribute information. Thus ACLs are only saved once.

The following Operating Systems are currently supported:

1. AIX (Extended Attributes)
2. Darwin (Extended Attributes)
3. FreeBSD (Extended Attributes)
4. IRIX (Extended Attributes)
5. Linux (Extended Attributes)
6. NetBSD (Extended Attributes)
7. Solaris (Extended Attributes and Extensible Attributes)
8. Tru64 (Extended Attributes)

ignore case=<yes|no> The default is no. On Windows systems, you will almost surely want to set this to **yes**. When this directive is set to **yes** all the case of character will be ignored in wild-card and regex comparisons. That is an uppercase A will match a lowercase a.

fstype=filesystem-type-list This option ensures that the FD, while processing files with this option block effective, will only descend into filesystems of types mentioned in the `Filesystem-type-list`. This is most useful in combination with `One FS = no` set.

This directive may appear multiple times, and all list elements will be added, i.e. a specification of

```
FS Type = ext2, xfs
FS Type = msdos
```

will result in all three mentioned file system types being accepted.

On Windows, this functionality is not available, but see "Drive Type" above (22.7).

DriveType=Windows-drive-type This option is effective only on Windows machines and is somewhat similar to the Unix/Linux **fstype** described above, except that it allows you to select what Windows drive types you want to allow. By default all drive types are accepted.

The permitted drivetype names are:

removable, fixed, remote, cdrom, ramdisk

You may have multiple Driveype directives, and thus permit matching of multiple drive types within a single Options resource. If the type specified on the drivetype directive does not match the filesystem for a particular directive, that directory will not be backed up. This directive can be used to prevent backing up non-local filesystems. Normally, when you use this directive, you would also set **onefs=no** so that Bacula will traverse filesystems.

This option is not implemented in Unix/Linux systems.

hfsplussupport=<yes|no> This option allows you to turn on support for Mac OSX HFS plus finder information.

strippath=<integer> This option will cause **integer** paths to be stripped from the front of the full path/filename being backed up. This can be useful if you are migrating data from another vendor or if you have taken a snapshot into some subdirectory. This directive can cause your filenames to be overlayed with regular backup data, so should be used only by experts and with great care.

<**file-list**> is a list of directory and/or filename names specified with a **File =** directive. To include names containing spaces, enclose the name between double-quotes. Wild-cards are not interpreted in file-lists. They can only be specified in Options resources.

There are a number of special cases when specifying directories and files in a **file-list**. They are:

- Any name preceded by an at-sign (@) is assumed to be the name of a file, which contains a list of files each preceded by a "File =". The named file is read once when the configuration file is parsed during the Director startup. Note, that the file is read on the Director's machine and not on the Client's. In fact, the @filename can appear anywhere within the conf file where a token would be read, and the contents of the named file will be logically inserted in the place of the @filename. What must be in the file depends on the location the @filename is specified in the conf file. For example:

```
Include {
  Options {compression=GZIP }
  @/home/files/my-files
}
```

- Any name beginning with a vertical bar (|) is assumed to be the name of a program. This program will be executed on the Director's machine at the time the Job starts (not when the Director reads the configuration file), and any output from that program will be assumed to be a list of files or directories, one per line, to be included. Before submitting the specified command bacula will performe character substitution.

  This allows you to have a job that, for example, includes all the local partitions even if you change the partitioning by adding a disk. The examples below show you how to do this. However, please note two things:

  1. if you want the local filesystems, you probably should be using the new **fstype** directive, which was added in version 1.36.3 and set **onefs=no**.
  2. the exact syntax of the command needed in the examples below is very system dependent. For example, on recent Linux systems, you may need to add the -P option, on FreeBSD systems, the options will be different as well.

  In general, you will need to prefix your command or commands with a `sh -c` so that they are invoked by a shell. This will not be the case if you are invoking a script as in the

second example below. Also, you must take care to escape (precede with a \) wild-cards, shell character, and to ensure that any spaces in your command are escaped as well. If you use a single quotes (') within a double quote ("), Bacula will treat everything between the single quotes as one field so it will not be necessary to escape the spaces. In general, getting all the quotes and escapes correct is a real pain as you can see by the next example. As a consequence, it is often easier to put everything in a file and simply use the file name within Bacula. In that case the `sh -c` will not be necessary providing the first line of the file is **#!/bin/sh**.

As an example:

```
Include {
   Options {signature = SHA1 }
   File = "|sh -c 'df -l | grep \"^/dev/hd[ab]\" | grep -v \".*/tmp\" \
      | awk \"{print \\$6}\"'"
}
```

will produce a list of all the local partitions on a Red Hat Linux system. Note, the above line was split, but should normally be written on one line. Quoting is a real problem because you must quote for Bacula which consists of preceding every \ and every " with a \, and you must also quote for the shell command. In the end, it is probably easier just to execute a small file with:

```
Include {
  Options {
    signature=MD5
  }
  File = "|my_partitions"
}
```

where my_partitions has:

```
#!/bin/sh
df -l | grep "^/dev/hd[ab]" | grep -v ".*/tmp" \
      | awk "{print \$6}"
```

If the vertical bar (|) in front of my_partitions is preceded by a backslash as in \|, the program will be executed on the Client's machine instead of on the Director's machine. Please note that if the filename is given within quotes, you will need to use two slashes. An example, provided by John Donagher, that backs up all the local UFS partitions on a remote system is:

```
FileSet {
  Name = "All local partitions"
  Include {
    Options {signature=SHA1; onefs=yes; }
    File = "\\|bash -c \"df -klF ufs | tail +2 | awk '{print \$6}'\""
  }
}
```

The above requires two backslash characters after the double quote (one preserves the next one). If you are a Linux user, just change the **ufs** to **ext3** (or your preferred filesystem type), and you will be in business.

If you know what filesystems you have mounted on your system, e.g. for Red Hat Linux normally only ext2 and ext3, you can backup all local filesystems using something like:

```
Include {
   Options {signature = SHA1; onfs=no; fstype=ext2 }
   File = /
}
```

On Windows, the command is executed with "cmd /c" prefix, and it is recommended to keep the path of the scripts as simple as possible (without spaces for example). The exact escaping sequence of the string can be difficult to determine.

- Any file-list item preceded by a less-than sign (<) will be taken to be a file. This file will be read on the Director's machine (see below for doing it on the Client machine) at the time the Job starts, and the data will be assumed to be a list of directories or files, one

per line, to be included. The names should start in column 1 and should not be quoted even if they contain spaces. This feature allows you to modify the external file and change what will be saved without stopping and restarting Bacula as would be necessary if using the @ modifier noted above. For example:

```
Include {
  Options {signature = SHA1 }
  File = "</home/files/local-filelist"
}
```

If you precede the less-than sign (<) with a backslash as in \<, the file-list will be read on the Client machine instead of on the Director's machine. Please note that if the filename is given within quotes, you will need to use two slashes.

```
Include {
  Options {signature = SHA1 }
  File = "\\</home/xxx/filelist-on-client"
}
```

- If you explicitly specify a block device such as /dev/hda1, then Bacula (starting with version 1.28) will assume that this is a raw partition to be backed up. In this case, you are strongly urged to specify a **sparse=yes** include option, otherwise, you will save the whole partition rather than just the actual data that the partition contains. For example:

```
Include {
  Options {signature=MD5; sparse=yes }
  File = /dev/hd6
}
```

will backup the data in device /dev/hd6. Note, the /dev/hd6 must be the raw partition itself. Bacula will not back it up as a raw device if you specify a symbolic link to a raw device such as my be created by the LVM Snapshot utilities.

Ludovic Strappazon has pointed out that this feature can be used to backup a full Microsoft Windows disk. Simply boot into the system using a Linux Rescue disk, then load a statically linked Bacula as described in the Disaster Recovery Using Bacula chapter of this manual. Then save the whole disk partition. In the case of a disaster, you can then restore the desired partition by again booting with the rescue disk and doing a restore of the partition.

- If you explicitly specify a FIFO device name (created with mkfifo), and you add the option **readfifo=yes** as an option, Bacula will read the FIFO and back its data up to the Volume. For example:

```
Include {
  Options {
    signature=SHA1
    readfifo=yes
  }
  File = /home/abc/fifo
}
```

if /home/abc/fifo is a fifo device, Bacula will open the fifo, read it, and store all data thus obtained on the Volume. Please note, you must have a process on the system that is writing into the fifo, or Bacula will hang, and after one minute of waiting, Bacula will give up and go on to the next file. The data read can be anything since Bacula treats it as a stream.

This feature can be an excellent way to do a "hot" backup of a very large database. You can use the **RunBeforeJob** to create the fifo and to start a program that dynamically reads your database and writes it to the fifo. Bacula will then write it to the Volume. Be sure to read the readfifo section that gives a tip to ensure that the RunBeforeJob does not block Bacula.

During the restore operation, the inverse is true, after Bacula creates the fifo if there was any data stored with it (no need to explicitly list it or add any options), that data will be written back to the fifo. As a consequence, if any such FIFOs exist in the fileset to be restored, you must ensure that there is a reader program or Bacula will block, and after one minute, Bacula will time out the write to the fifo and move on to the next file.

- A file-list may not contain wild-cards. Use directives in the Options resource if you wish to specify wild-cards or regular expression matching.
- The **ExcludeDirContaining =** <**filename**> is a directive that can be added to the Include section of the FileSet resource. If the specified filename (**filename-string**) is found on the Client in any directory to be backed up, the whole directory will be ignored (not backed up). For example:

```
# List of files to be backed up
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
    }
    File = /home
    Exclude Dir Containing = .excludeme
  }
}
```

But in /home, there may be hundreds of directories of users and some people want to indicate that they don't want to have certain directories backed up. For example, with the above FileSet, if the user or sysadmin creates a file named .excludeme in specific directories, such as

```
/home/user/www/cache/.excludeme
/home/user/temp/.excludeme
```

then Bacula will not backup the two directories named:

```
/home/user/www/cache
/home/user/temp
```

NOTE: subdirectories will not be backed up. That is, the directive applies to the two directories in question and any children (be they files, directories, etc).

22.8 FileSet Examples

The following is an example of a valid FileSet resource definition. Note, the first Include pulls in the contents of the file /etc/backup.list when Bacula is started (i.e. the @), and that file must have each filename to be backed up preceded by a **File =** and on a separate line.

```
FileSet {
  Name = "Full Set"
  Include {
    Options {
      Compression=GZIP
      signature=SHA1
      Sparse = yes
    }
    @/etc/backup.list
  }
  Include {
     Options {
        wildfile = "*.o"
        wildfile = "*.exe"
        Exclude = yes
     }
     File = /root/myfile
     File = /usr/lib/another_file
  }
}
```

In the above example, all the files contained in /etc/backup.list will be compressed with GZIP compression, an SHA1 signature will be computed on the file's contents (its data), and sparse file handling will apply.

The two directories /root/myfile and /usr/lib/another_file will also be saved without any options, but all files in those directories with the extensions .o and .exe will be excluded.

Let's say that you now want to exclude the directory /tmp. The simplest way to do so is to add an **Exclude** directive that lists /tmp. The example above would then become:

```
FileSet {
  Name = "Full Set"
  Include {
    Options {
      Compression=GZIP
      signature=SHA1
      Sparse = yes
    }
    @/etc/backup.list
  }
  Include {
     Options {
        wildfile = "*.o"
        wildfile = "*.exe"
        Exclude = yes
     }
     File = /root/myfile
     File = /usr/lib/another_file
  }
  Exclude {
     File = /tmp                          # don't add trailing /
  }
}
```

You can add wild-cards to the File directives listed in the Exclude directive, but you need to take care because if you exclude a directory, it and all files and directories below it will also be excluded.

Now let's make a slight variation on the above and suppose you want to save all your filesystems except /tmp. The problem that comes up is that Bacula will not normally cross from one filesystem to another. Doing a df command, you get the following output:

```
[kern@rufus k]$ df
Filesystem      1k-blocks      Used Available Use% Mounted on
/dev/hda5         5044156    439232   4348692  10% /
/dev/hda1           62193      4935     54047   9% /boot
/dev/hda9        20161172   5524660  13612372  29% /home
/dev/hda2           62217      6843     52161  12% /rescue
/dev/hda8         5044156     42548   4745376   1% /tmp
/dev/hda6         5044156   2613132   2174792  55% /usr
none               127708         0    127708   0% /dev/shm
//minimatou/c$   14099200   9895424   4203776  71% /mnt/mmatou
lmatou:/          1554264    215884   1258056  15% /mnt/matou
lmatou:/home      2478140   1589952    760072  68% /mnt/matou/home
lmatou:/usr       1981000   1199960    678628  64% /mnt/matou/usr
lpmatou:/          995116    484112    459596  52% /mnt/pmatou
lpmatou:/home    19222656   2787880  15458228  16% /mnt/pmatou/home
lpmatou:/usr      2478140   2038764    311260  87% /mnt/pmatou/usr
deuter:/          4806936     97684   4465064   3% /mnt/deuter
deuter:/home      4806904    280100   4282620   7% /mnt/deuter/home
deuter:/files    44133352  27652876  14238608  67% /mnt/deuter/files
```

And we see that there are a number of separate filesystems (/ /boot /home /rescue /tmp and /usr not to mention mounted systems). If you specify only / in your Include list, Bacula will only save the Filesystem /dev/hda5. To save all filesystems except /tmp with out including any of the Samba or NFS mounted systems, and explicitly excluding a /tmp, /proc, .journal, and .autofsck, which you will not want to be saved and restored, you can use the following:

```
FileSet {
  Name = Include_example
```

```
  Include {
    Options {
       wilddir = /proc
       wilddir = /tmp
       wildfile = "/.journal"
       wildfile = "/.autofsck"
       exclude = yes
    }
    File = /
    File = /boot
    File = /home
    File = /rescue
    File = /usr
  }
}
```

Since /tmp is on its own filesystem and it was not explicitly named in the Include list, it is not really needed in the exclude list. It is better to list it in the Exclude list for clarity, and in case the disks are changed so that it is no longer in its own partition.

Now, lets assume you only want to backup .Z and .gz files and nothing else. This is a bit trickier because Bacula by default will select everything to backup, so we must exclude everything but .Z and .gz files. If we take the first example above and make the obvious modifications to it, we might come up with a FileSet that looks like this:

```
FileSet {
  Name = "Full Set"
  Include {                    !!!!!!!!!!!!
     Options {                    This
        wildfile = "*.Z"           example
        wildfile = "*.gz"          doesn't
                                   work
     }                           !!!!!!!!!!!!
     File = /myfile
  }
}
```

The *.Z and *.gz files will indeed be backed up, but all other files that are not matched by the Options directives will automatically be backed up too (i.e. that is the default rule).

To accomplish what we want, we must explicitly exclude all other files. We do this with the following:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wildfile = "*.Z"
        wildfile = "*.gz"
     }
     Options {
        Exclude = yes
        RegexFile = ".*"
     }
     File = /myfile
  }
}
```

The "trick" here was to add a RegexFile expression that matches all files. It does not match directory names, so all directories in /myfile will be backed up (the directory entry) and any *.Z and *.gz files contained in them. If you know that certain directories do not contain any *.Z or *.gz files and you do not want the directory entries backed up, you will need to explicitly exclude those directories. Backing up a directory entry is not very expensive.

Bacula uses the system regex library and some of them are different on different OSes. The above has been reported not to work on FreeBSD. This can be tested by using the estimate

job=job-name listing command in the console and adapting the RegexFile expression appropriately. In a future version of Bacula, we will supply our own Regex code to avoid such system dependencies.

Please be aware that allowing Bacula to traverse or change file systems can be **very** dangerous. For example, with the following:

```
FileSet {
  Name = "Bad example"
  Include {
    Options {onefs=no }
    File = /mnt/matou
  }
}
```

you will be backing up an NFS mounted partition (/mnt/matou), and since **onefs** is set to **no**, Bacula will traverse file systems. Now if /mnt/matou has the current machine's file systems mounted, as is often the case, you will get yourself into a recursive loop and the backup will never end.

As a final example, let's say that you have only one or two subdirectories of /home that you want to backup. For example, you want to backup only subdirectories beginning with the letter a and the letter b – i.e. /home/a* and /home/b*. Now, you might first try:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wilddir = "/home/a*"
        wilddir = "/home/b*"
     }
     File = /home
  }
}
```

The problem is that the above will include everything in /home. To get things to work correctly, you need to start with the idea of exclusion instead of inclusion. So, you could simply exclude all directories except the two you want to use:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        RegexDir = "^/home/[c-z]"
        exclude = yes
     }
     File = /home
  }
}
```

And assuming that all subdirectories start with a lowercase letter, this would work.

An alternative would be to include the two subdirectories desired and exclude everything else:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wilddir = "/home/a*"
        wilddir = "/home/b*"
     }
     Options {
        RegexDir = ".*"
        exclude = yes
     }
```

```
      File = /home
    }
}
```

The following example shows how to back up only the My Pictures directory inside the My Documents directory for all users in C:/Documents and Settings, i.e. everything matching the pattern:

C:/Documents and Settings/*/My Documents/My Pictures/*

To understand how this can be achieved, there are two important points to remember:

Firstly, Bacula traverses the filesystem starting from the File = lines. It stops descending when a directory is excluded, so you must include all ancestor (higher level) directories of each directory containing files to be included.

Secondly, each directory and file is compared to the Options clauses in the order they appear in the FileSet. When a match is found, no further Optionss are compared and the directory or file is either included or excluded.

The FileSet resource definition below implements this by including specifc directories and files and excluding everything else.

```
FileSet {
  Name = "AllPictures"

  Include {

    File  = "C:/Documents and Settings"

    Options {
      signature = SHA1
      verify = s1
      IgnoreCase = yes

      # Include all users' directories so we reach the inner ones.  Unlike a
      # WildDir pattern ending in *, this RegExDir only matches the top-level
      # directories and not any inner ones.
      RegExDir = "^C:/Documents and Settings/[^/]+$"

      # Ditto all users' My Documents directories.
      WildDir = "C:/Documents and Settings/*/My Documents"

      # Ditto all users' My Documents/My Pictures directories.
      WildDir = "C:/Documents and Settings/*/My Documents/My Pictures"

      # Include the contents of the My Documents/My Pictures directories and
      # any subdirectories.
      Wild = "C:/Documents and Settings/*/My Documents/My Pictures/*"
    }

    Options {
      Exclude = yes
      IgnoreCase = yes

      # Exclude everything else, in particular any files at the top level and
      # any other directories or files in the users' directories.
      Wild = "C:/Documents and Settings/*"
    }
  }
}
```

In the above example, we setup a FileSet to backup a subset of a single directory (on Windows) based on filename.

```
FileSet {
  Name = "win-fileset"
```

```
  Include {
   Options {
     signature = MD5
     wildfile = "C:/File/backup_NDB_COUNT*"
   }
   # Exclude all files not matching the wildfile list above
   Options {
     Exclude = yes
     RegexFile = ".*"
   }
   File = C:/File
 }
}
```

22.9 Backing up Raw Partitions

The following FileSet definition will backup a raw partition:

```
FileSet {
  Name = "RawPartition"
  Include {
    Options {sparse=yes }
    File = /dev/hda2
  }
}
```

While backing up and restoring a raw partition, you should ensure that no other process including the system is writing to that partition. As a precaution, you are strongly urged to ensure that the raw partition is not mounted or is mounted read-only. If necessary, this can be done using the **RunBeforeJob** directive.

22.10 Excluding Files and Directories

You may also include full filenames or directory names in addition to using wild-cards and **Exclude=yes** in the Options resource as specified above by simply including the files to be excluded in an Exclude resource within the FileSet. It accepts wild-cards pattern, so for a directory, don't add a trailing /. For example:

```
FileSet {
  Name = Exclusion_example
  Include {
    Options {
      Signature = SHA1
    }
    File = /
    File = /boot
    File = /home
    File = /rescue
    File = /usr
  }
  Exclude {
    File = /proc
    File = /tmp                          # Don't add trailing /
    File = .journal
    File = .autofsck
  }
}
```

22.11 Windows FileSets

If you are entering Windows file names, the directory path may be preceded by the drive and a colon (as in c:). However, the path separators must be specified in Unix convention (i.e. forward slash (/)). If you wish to include a quote in a file name, precede the quote with a backslash (\). For example you might use the following for a Windows machine to backup the "My Documents" directory:

```
FileSet {
  Name = "Windows Set"
  Include {
    Options {
       WildFile = "*.obj"
       WildFile = "*.exe"
       exclude = yes
     }
     File = "c:/My Documents"
  }
}
```

For exclude lists to work correctly on Windows, you must observe the following rules:

- Filenames are case sensitive, so you must use the correct case.
- To exclude a directory, you must not have a trailing slash on the directory name.
- If you have spaces in your filename, you must enclose the entire name in double-quote characters ("). Trying to use a backslash before the space will not work.
- If you are using the old Exclude syntax (noted below), you may not specify a drive letter in the exclude. The new syntax noted above should work fine including driver letters.

Thanks to Thiago Lima for summarizing the above items for us. If you are having difficulties getting includes or excludes to work, you might want to try using the `estimate job=xxx listing` command documented in the **estimate** command (command 1.5 page 7) of Bacula Community Edition Console manual.

On Win32 systems, if you move a directory or file or rename a file into the set of files being backed up, and a Full backup has already been made, Bacula will not know there are new files to be saved during an Incremental or Differential backup (blame Microsoft, not me). To avoid this problem, please **copy** any new directory or files into the backup area. If you do not have enough disk to copy the directory or files, move them, but then initiate a Full backup.

A Windows Example FileSet The following example was contributed by Russell Howe. Please note that for presentation purposes, the lines beginning with Data and Internet have been wrapped and should included on the previous line with one space.

```
This is my Windows 2000 fileset:
FileSet {
 Name = "Windows 2000"
 Include {
  Options {
   signature = MD5
   Exclude = yes
   IgnoreCase = yes
   # Exclude Mozilla-based programs' file caches
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/Cache"
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/Cache.Trash"
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/ImapMail"
```

```
   # Exclude user's registry files - they're always in use anyway.
   WildFile = "[A-Z]:/Documents and Settings/*/Local Settings/Application
Data/Microsoft/Windows/usrclass.*"
   WildFile = "[A-Z]:/Documents and Settings/*/ntuser.*"

   # Exclude directories full of lots and lots of useless little files
   WildDir = "[A-Z]:/Documents and Settings/*/Cookies"
   WildDir = "[A-Z]:/Documents and Settings/*/Recent"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/History"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/Temp"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/Temporary
Internet Files"

   # These are always open and unable to be backed up
   WildFile = "[A-Z]:/Documents and Settings/All Users/Application
Data/Microsoft/Network/Downloader/qmgr[01].dat"

   # Some random bits of Windows we want to ignore
   WildFile = "[A-Z]:/WINNT/security/logs/scepol.log"
   WildDir = "[A-Z]:/WINNT/system32/config"
   WildDir = "[A-Z]:/WINNT/msdownld.tmp"
   WildDir = "[A-Z]:/WINNT/Internet Logs"
   WildDir = "[A-Z]:/WINNT/$Nt*Uninstall*"
   WildDir = "[A-Z]:/WINNT/sysvol"
   WildFile = "[A-Z]:/WINNT/cluster/CLUSDB"
   WildFile = "[A-Z]:/WINNT/cluster/CLUSDB.LOG"
   WildFile = "[A-Z]:/WINNT/NTDS/edb.log"
   WildFile = "[A-Z]:/WINNT/NTDS/ntds.dit"
   WildFile = "[A-Z]:/WINNT/NTDS/temp.edb"
   WildFile = "[A-Z]:/WINNT/ntfrs/jet/log/edb.log"
   WildFile = "[A-Z]:/WINNT/ntfrs/jet/ntfrs.jdb"
   WildFile = "[A-Z]:/WINNT/ntfrs/jet/temp/tmp.edb"
   WildFile = "[A-Z]:/WINNT/system32/CPL.CFG"
   WildFile = "[A-Z]:/WINNT/system32/dhcp/dhcp.mdb"
   WildFile = "[A-Z]:/WINNT/system32/dhcp/j50.log"
   WildFile = "[A-Z]:/WINNT/system32/dhcp/tmp.edb"
   WildFile = "[A-Z]:/WINNT/system32/LServer/edb.log"
   WildFile = "[A-Z]:/WINNT/system32/LServer/TLSLic.edb"
   WildFile = "[A-Z]:/WINNT/system32/LServer/tmp.edb"
   WildFile = "[A-Z]:/WINNT/system32/wins/j50.log"
   WildFile = "[A-Z]:/WINNT/system32/wins/wins.mdb"
   WildFile = "[A-Z]:/WINNT/system32/wins/winstmp.mdb"

   # Temporary directories & files
   WildDir = "[A-Z]:/WINNT/Temp"
   WildDir = "[A-Z]:/temp"
   WildFile = "*.tmp"
   WildDir = "[A-Z]:/tmp"
   WildDir = "[A-Z]:/var/tmp"

   # Recycle bins
   WildDir = "[A-Z]:/RECYCLER"

   # Swap files
   WildFile = "[A-Z]:/pagefile.sys"

   # These are programs and are easier to reinstall than restore from
   # backup
   WildDir = "[A-Z]:/cygwin"
   WildDir = "[A-Z]:/Program Files/Grisoft"
   WildDir = "[A-Z]:/Program Files/Java"
   WildDir = "[A-Z]:/Program Files/Java Web Start"
   WildDir = "[A-Z]:/Program Files/JavaSoft"
   WildDir = "[A-Z]:/Program Files/Microsoft Office"
   WildDir = "[A-Z]:/Program Files/Mozilla Firefox"
   WildDir = "[A-Z]:/Program Files/Mozilla Thunderbird"
   WildDir = "[A-Z]:/Program Files/mozilla.org"
   WildDir = "[A-Z]:/Program Files/OpenOffice*"
  }

  # Our Win2k boxen all have C: and D: as the main hard drives.
  File = "C:/"
  File = "D:/"
 }
}
```

Note, the three line of the above Exclude were split to fit on the document page, they should be written on a single line in real use.

Windows NTFS Naming Considerations NTFS filenames containing Unicode characters should now be supported as of version 1.37.30 or later.

22.12 Testing Your FileSet

If you wish to get an idea of what your FileSet will really backup or if your exclusion rules will work correctly, you can test it by using the estimate command in the Console program. See the estimate command (command 1.5 page 7) of Bacula Community Edition Console manual.

As an example, suppose you add the following test FileSet:

```
FileSet {
  Name = Test
  Include {
    File = /home/xxx/test
    Options {
       regex = ".*\.c$"
    }
  }
}
```

You could then add some test files to the directory /home/xxx/test and use the following command in the console:

```
estimate job=<any-job-name> listing client=<desired-client> fileset=Test
```

to give you a listing of all files that match. In the above example, it should be only files with names ending in .c.

22.12.1 Include All Windows Drives in FileSet

The alldrives Windows Plugin allows you to include all local drives with a simple directive. This plugin is available in the Windows 64 and 32 bit installer.

```
FileSet {
 Name = EverythingFS
 ...
 Include {
   Plugin = "alldrives"
 }
}
```

You exclude some specific drives with the exclude option.

```
FileSet {
 Name = EverythingFS
 ...
 Include {
   Plugin = "alldrives: exclude=D,E"
 }
}
```

The alldrives Windows plugin only considers regular drives (i.e. C:/, D:/, E:/). Mount points inside the directory tree will no be part of the snapshots unless you set "**OneFS = no**" in the Options block of the FileSet, which will cause Bacula to snapshot everything.

22.12.2 Microsoft VSS Writer Plugin

We provide a single plugin named vss-fd.dll that permits you to backup a number of different components on Windows machines. This plugin is available from Bacula Systems as an option.

Please write to the Support Team[4] to find more information about our VSS products.

- System State writers
  - Registry
  - Event Logs
  - COM+ REGDB (COM Registration Database)
  - System (Systems files – most of what is under c:/windows and more)
  - Windows Management and Instrumentation (WMI)
  - NT Directory Service (NTDS) (Active Directory)
  - NT File Replication Service (NTFRS) (SYSVOL etc replication – Windows 2003 domains)
  - Distributed File System (DFS) Replication (SYSVOLS etc replication – Windows 2008 domains)
  - Automated System Recovery (ASR) Writer

  This component is known to work.
- MSSQL databases (except those owned by Sharepoint if that plugin is specified).
  This component has been tested, but only works for Full backups. Please do not attempt to use it for incremental backups. The Windows writer performs block level delta for Incremental backups, which are only supported by Bacula version 4.2.0, not yet released. If you use this component, please do not use it in production without careful testing.
- Exchange (all exchange databases)
  We have tested this component and found it to work, but only for Full backups. Please do not attempt to use it for incremental or differential backups. We are including this component for you to test. Please do not use it in production without careful testing.
  Bacula Systems has a White Paper that describes backup and restore of MS Exchange 2010 in detail.

Each of the above specified Microsoft components can be backed up by specifying a different plugin option within the Bacula FileSet. All specifications must start with **vss:** and be followed with a keyword which indicates the writer, such as **/@SYSTEMSTATE/** (see below). To activate each component you use the following:

- System State writers

  ```
  Plugin = "vss:/@SYSTEMSTATE/"
  ```

  Note, exactly which subcomponents will be backed up depends on which ones you have enabled within Windows. For example, on a standard default Vista system only ASR Writer, COM+ REGDB, System State, and WMI are enabled.
- MSSQL databases (except those owned by Sharepoint if that plugin is specified)

  ```
  Plugin = "vss:/@MSSQL/"
  ```

  The Microsoft literature says that the mssql writer is only good for snapshots and it needs to be enabled via a registry tweak or else the older MSDE writer will be invoked instead.
- Exchange (all exchange databases)

  ```
  Plugin = "vss:/@EXCHANGE/"
  ```

[4] mailto:support@baculasystems.com

The plugin directives must be specified exactly as shown above. A Job may have one or more of the **vss** plugins components specified.

Also ensure that the vss-fd.dll plugin is in the plugins directory on the FD doing the backup, and that the plugin directory config line is present in the FD's configuration file (bacula-fd.conf).

Backup

If everything is set up correctly as above then the backup should include the system state. The system state files backed up will appear in a bconsole or BAT restore like:

```
/@SYSTEMSTATE/
/@SYSTEMSTATE/ASR Writer/
/@SYSTEMSTATE/COM+ REGDB Writer/
```

etc

Only a complete backup of the system state is supported at this time. That is it is not currently possible to just back up the Registry or Active Directory by itself. In almost all cases a complete backup is a good idea anyway as most of the components are interconnected in some way. Also, if an incremental or differential backup is specified on the backup Job then a full backup of the system state will still be done. The size varies according to your installation. We have seen up to 6GB under Windows 2008, mostly because of the "System" writer, and up to 20GB on Vista. The actual size depends on how many Windows components are enabled.

The system state component automatically respects all the excludes present in the FilesNotToBackup registry key, which includes things like %TEMP%, pagefile.sys, hiberfil.sys, etc. Each plugin may additionally specify files to exclude, eg the VSS Registry Writer will tell Bacula to not back up the registry hives under C:\WINDOWS\system32\config because they are backed up as part of the system state.

Restore

In most cases a restore of the entire backed up system state is recommended. Individual writers can be selected for restore, but currently not individual components of those writers. To restore just the Registry, you would need to mark @SYSTEMSTATE (only the directory, not the subdirectories), and then do **mark Registry*** to mark the Registry writer and everything under it.

Restoring anything less than a single component may not produce the intended results and should only be done if a specific need arises and you know what you are doing, and not without testing on a non-critical system first.

To restore Active Directory, the system will need to be booted into Directory Services Restore Mode, an option at Windows boot time.

Only a non-authoritative restore of NTFRS/DFSR is supported at this time. There exists Windows literature to turn a Domain Controller (DC) restored in non-authoritative mode back into an authoritative Domain Controller. If only one DC exists it appears that Windows does an authoritative restore anyway.

Most VSS components will want to restore to files that are currently in use. A reboot will be required to complete the restore (eg to bring the restored registry online).

Starting another restore of VSS data after the restore of the registry without first rebooting will not produce the intended results as the "to be replaced next reboot" file list will only be updated in the "to be replaced" copy of the registry and so will not be actioned.

Example

Suppose you have the following backup FileSet:

```
@SYSTEMSTATE/
  System Writer/
    instance_{GUID}
    System Files/
  Registry Writer/
    instance_{GUID}
    Registry/
  COM+ REGDB Writer/
    instance_{GUID}
    COM+ REGDB/
  NTDS/
    instance_{GUID}
    ntds/
```

If only the Registry needs to be restored, then you could use the following commands in bconsole:

```
markdir @SYSTEMSTATE
cd @SYSTEMSTATE
markdir "Registry Writer"
cd "Registry Writer"
mark instance*
mark "Registry"
```

Windows Plugins Items to Note

- Reboot Required after a Plugin Restore
  In general after any VSS plugin is used to restore a component, you will need to reboot the system. This is required because in-use files cannot be replaced during restore time, so they are noted in the registry and replaced when the system reboots.
- After a System State restore, a reboot will generally take longer than normal because the pre-boot process must move the newly restored files into their final place prior to actually booting the OS.
- One File from Each Drive needed by the Plugins must be backed up
  At least one file from each drive that will be needed by the plugin must have a regular file that is marked for backup. This is to ensure that the main Bacula code does a snapshot of all the required drives. At a later time, we will find a way to accomplish this automatically.
- Bacula does not Automatically Backup Mounted Drives
  Any drive that is mounted in the normal file structure using a mount point or junction point will not be backed up by Bacula. If you want it backed up, you must explicitly mention it in a Bacula "**File**" directive in your FileSet.
- When doing a backup that is to be used as a Bare Metal Recovery, do not use the VSS plugin. The reason is that during a Bare Metal Recovery, VSS is not available nor are the writers from the various components that are needed to do the restore. You might do full backup to be used with a Bare Metal Recovery once a month or once a week, and all other days, do a backup using the VSS plugin, but under a different Job name. Then to restore your system, use the last Full non-VSS backup to restore your system, and after rebooting do a restore with the VSS plugin to get everything fully up to date.

This plugin is available as an option. Please contact Bacula Systems to get access to the VSS Plugin packages and the documentation.

22.12.3 Support for NDMP Protocol

The new ndmp Plugin is able to backup a NAS through NDMP protocol using **Filer to server** approach, where the Filer is backing up across the LAN to your Bacula server.

Accurate option should be turned on in the Job resource.

```
Job {
 Accurate = yes
 FileSet = NDMPFS
 ...
}

FileSet {
 Name = NDMPFS
 ...
 Include {
   Plugin = "ndmp:host=nasbox user=root pass=root file=/vol/vol1"
 }
}
```

This plugin is available as an option. Please contact Bacula Systems to get access to the NDMP Plugin packages and the documentation.

22.12.4 Incremental Accelerator Plugin for NetApp

The Incremental Accelerator for NetApp Plugin is designed to simplify the backup and restore procedure of your NetApp NAS hosting a huge number of files.

When using the NetApp HFC Plugin, Bacula Enterprise will query the NetApp device to get the list of all files modified since the last backup instead of having to walk through the entire filesystem. Once Bacula have the list of all files to back's up, it will use a standard network share (such as NFS or CIFS) to access files.

This plugin is available as an option. Please contact Bacula Systems to get access to the Incremental Accelerator Plugin for NetApp packages and the documentation.

22.12.5 PostgreSQL Plugin

The PostgreSQL plugin is designed to simplify the backup and restore procedure of your PostgreSQL cluster, the backup administrator doesn't need to learn about internals of PostgreSQL backup techniques or write complex scripts. The plugin will automatically take care for you to backup essential information such as configuration, users definition or tablespaces. The PostgreSQL plugin supports both dump and Point In Time Recovery (PITR) backup techniques.

This plugin is available as an option. Please contact Bacula Systems to get access to the PostgreSQL Plugin packages and the documentation.

22.12.6 VMWare vSphere VADP Plugin

The Bacula Enterprise vSphere plugin provides virtual machine bare metal recovery, while the backup at the guest level simplify data protection of critical applications.

The plugin integrates the VMware's Changed Block Tracking (CBT) technology to ensure only blocks that have changed since the initial Full, and/or the last Incremental or Differential Backup are sent to the current Incremental or Differential backup stream to give you more efficient backups and reduced network load.

This plugin is available as an option. Please contact Bacula Systems to get access to the vSphere Plugin packages and the documentation.

22.12.7 Oracle RMAN Plugin

The Bacula Enterprise Oracle Plugin is designed to simplify the backup and restore procedure of your Oracle Database instance, the backup administrator don't need to learn about internals of Oracle backup techniques or write complex scripts. The Bacula Enterprise Oracle plugin supports both dump and PITR with Oracle Recovery Manager (RMAN) backup techniques.

This plugin is available as an option. Please contact Bacula Systems to get access to the Oracle Plugin packages and the documentation.

22.13 The Client Resource

The Client resource defines the attributes of the Clients that are served by this Director; that is the machines that are to be backed up. You will need one Client resource definition for each machine to be backed up.

Client (or FileDaemon) Start of the Client directives.

Name = <name> The client name which will be used in the Job resource directive or in the console `run` command. This directive is required.

Enabled = <yes|no> This directive allows you to enable or disable the `Client` resource. If the resource is disabled, the Client will not be used.

Address = <address> Where the <address> is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula File server daemon. This directive is required.

FD Port = <port-number> Where the <port-number> is a port number at which the Bacula File server daemon can be contacted. The default is 9102.

AllowFDConnections = <yes|no> When **AllowFDConnections** is set to true, the Director will accept incoming connections from the Client and will keep the socket open for a future use. The Director will no longer use the **Address** to contact the File Daemon. This configuration is useful if the Director cannot contact the FileDaemon directly. See 23.2 on page 307 for more information. The default value is no.

Catalog = <Catalog-resource-name> This specifies the name of the catalog resource to be used for this Client. This directive is required.

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This directive is required. If you have either `/dev/random` or `bc` on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process, but it is preferable for security reasons to make the text random.

Snapshot Retention = <time-period-specification> The Snapshot Retention directive defines the length of time that Bacula will keep Snapshots in the Catalog database and on the Client after the Snapshot creation. When this time period expires, and if using the `snapshot prune` command, Bacula will prune (remove) Snapshot records that are older than the specified Snapshot Retention period and will contact the FileDaemon to delete Snapshots from the system.

The Snapshot retention period is specified as seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 0 seconds, Snapshots are deleted at the end of the backup. The Job **SnapshotRetention** directive overwrites the Client **SnapshotRetention** directive.

File Retention = <time-period-specification> The File Retention directive defines the length of time that Bacula will keep File records in the Catalog database after the End time of the Job corresponding to the File records. When this time period expires, and if **AutoPrune** is set to yes Bacula will prune (remove) File records that are older than the specified File Retention period. Note, this affects only records in the catalog database. It does not affect your archive backups.

File records may actually be retained for a shorter period than you specify on this directive if you specify either a shorter **Job Retention** or a shorter **Volume Retention** period. The shortest retention period of the three takes precedence. The time may be expressed in seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 60 days.

Job Retention = <time-period-specification> The Job Retention directive defines the length of time that Bacula will keep Job records in the Catalog database after the Job End time. When this time period expires, and if **AutoPrune** is set to yes Bacula will prune (remove) Job records that are older than the specified File Retention period. As with the other retention periods, this affects only records in the catalog and not data in your archive backup.

If a Job record is selected for pruning, all associated File and JobMedia records will also be pruned regardless of the File Retention period set. As a consequence, you normally will set the File retention period to be less than the Job retention period. The Job retention period can actually be less than the value you specify here if you set the **Volume Retention** directive in the Pool resource to a smaller duration. This is because the Job retention period and the Volume retention period are independently applied, so the smaller of the two takes precedence.

The Job retention period is specified as seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 180 days.

AutoPrune = <yes|no> If AutoPrune is set to yes (default), Bacula (version 1.20 or greater) will automatically apply the File retention period and the Job retention period for the Client at the end of the Job. If you set **AutoPrune**=no, pruning will not be done, and your Catalog will grow in size each time you run a Job. Pruning affects only information in the catalog and not data stored in the backup archives (on Volumes).

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs with the current Client that can run concurrently. Note, this directive limits only Jobs for Clients with the same name as the resource in which it appears. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Storage resources will also apply in addition to any limit specified here. The default is set to 1, but you may set it to a larger number. If set to a large value, please be careful to not have this value higher than the **Maximum Concurrent Jobs** configured in the `Client` resource in the Client/File daemon configuration file. Otherwise, backup jobs can fail due to the Director connection to FD be refused because Maximum Concurrent Jobs was exceeded on FD side.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes that a job may use when started for this Client. You may specify the following speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfer with the value set with the Directive.

Priority = <number> The number specifies the priority of this client relative to other clients that the Director is processing simultaneously. The priority can range from 1 to 1000. The clients are ordered such that the smaller number priorities are performed first (not currently implemented).

SD Calls Client = <yes|no> If the **SD Calls Client** directive is set to true in a `Client` resource any Backup, Restore, Verify Job where the client is involved, the client will wait for the Storage daemon to contact it. By default this directive is set to false, and the Client will call the Storage daemon as it always has. This directive can be useful if your Storage daemon is behind a firewall that permits outgoing connections but not incoming connections.

FD Storage Address = <address> Where the <address> is a host name, a **FQDN!**, or an **IP address**. The <address> specified here will be transmitted to the File daemon instead of the IP address that the Director uses to contact the Storage daemon. This FDStorageAddress will then be used by the File daemon to contact the Storage daemon. This directive particularly useful if the File daemon is in a different network domain than the Director or Storage daemon. It is also useful in NAT or firewal environments.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
```

```
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[5] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
```

[5] https://en.wikipedia.org/wiki/X.509

```
    ...
    # TLS configuration directives
    TLS Enable = yes
    TLS Require = yes
    # the Allowed CN will be checked for this client by director
    # the client's certificate Common Name must match any of
    # the values of the Allowed CN list
    TLS Allowed CN = client1.example.com
    TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
    TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
    TLS Key = /opt/bacula/ssl/keys/director_key.pem
  }
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following is an example of a valid Client resource definition:

```
Client {
  Name = Minimatou
  Address = minimatou
  Catalog = MySQL
  Password = very_good
}
```

22.14 The Storage Resources

The Storage resource defines which Storage daemons are available for use by the Director.

Storage Start of the Storage resources. At least one storage resource must be specified.

Name = <name> The name of the storage resource. This name appears on the Storage directive specified in the Job resource and is required.

Enabled = <yes|no> This directive allows you to enable or disable a `Storage` resource. When the resource is disabled, the storage device will not be used. To reuse it you must re-enable the `Storage` resource.

Address = <address> Where the address is a host name, a **FQDN!**, or an **IP address**. Please note that the <address> as specified here will be transmitted to the File daemon who will then use it to contact the Storage daemon. Hence, it is **not**, a good idea to use localhost as the name but rather a fully qualified machine name or an IP address. This directive is required.

FD Storage Address = <address> Where the <address> is a host name, a **FQDN!**, or an **IP address**. The <address> specified here will be transmitted to the File daemon instead of the IP address that the Director uses to contact the Storage daemon. This FDStorageAddress will then be used by the File daemon to contact the Storage daemon. This directive particularly useful if the File daemon is in a different network domain than the Director or Storage daemon. It is also useful in NAT or firewal environments.

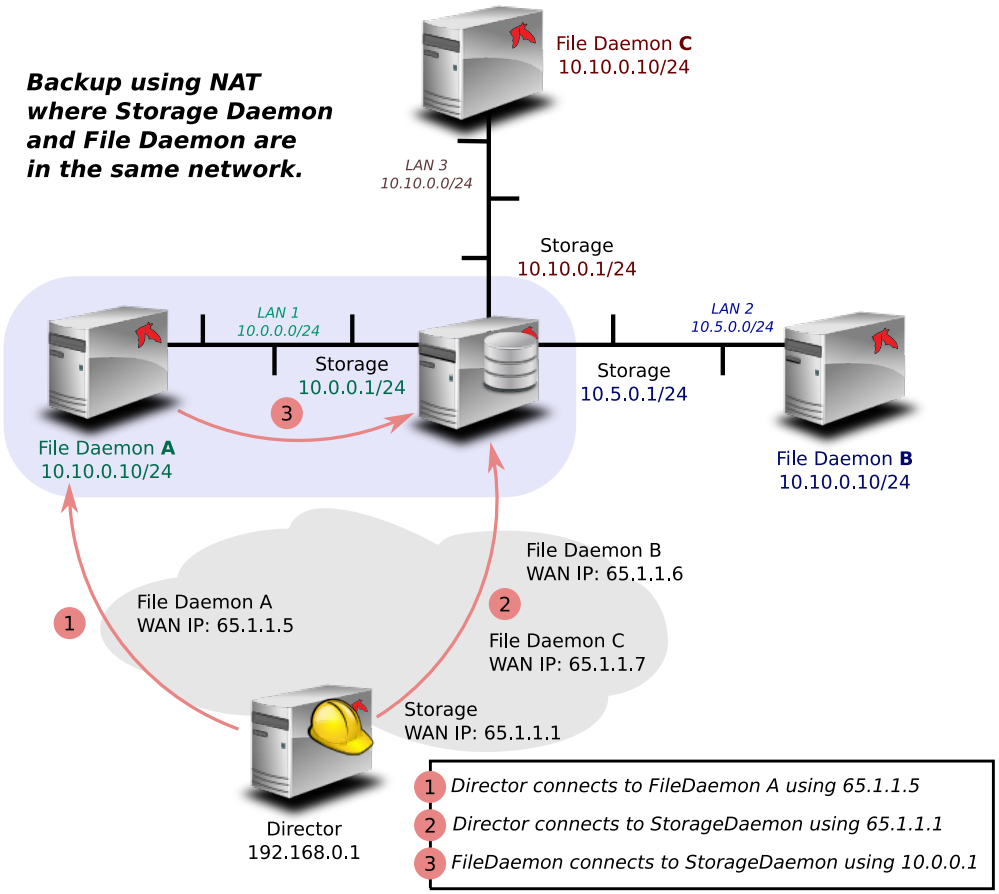


Figure 22.3: Backup over WAN using FD Storage Address

SD Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is 9103.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This directive is required. If you have either `/dev/random` or `bc` on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process, but it is preferable for security reasons to use random text.

Device = <device-name> This directive specifies the Storage daemon's name of the device resource to be used for the storage. If you are using an Autochanger, the name specified here should be the name of the Storage daemon's Autochanger resource rather than the name of an individual device. This name is not the physical device name, but the logical device name as defined on the **Name** directive contained in the `Device` or the `Autochanger` resource definition of the **Storage daemon** configuration file. You can specify any name you would like (even the device name if you prefer) up to a maximum of 127 characters in length. The physical device name associated with this device is specified in the **Storage daemon** configuration file (as **Archive Device**). Please take care not to define two different Storage resource directives in the Director that point to the same Device in the Storage daemon. Doing so may cause the Storage daemon to block (or hang) attempting to open the same device that is already open. This directive is required.

Media Type = <MediaType> This directive specifies the Media Type to be used to store the data. This is an arbitrary string of characters up to 127 maximum that you define. It can be anything you want. However, it is best to make it descriptive of the storage media (e.g. "File", "DAT", "HP DLT8000", "8mm", ...). In addition, it is essential that you make the **Media Type** specification unique for each storage media type. If you have two DDS-4 drives that have incompatible formats, or if you have a DDS-4 drive and a DDS-4 autochanger, you almost certainly should specify different **Media Types**. During a restore, assuming a **DDS-4** Media Type is associated with the Job, Bacula can decide to use any Storage daemon that supports Media Type **DDS-4** and on any drive that supports it.

If you are writing to disk Volumes, you must make doubly sure that each Device resource defined in the Storage daemon (and hence in the Director's conf file) has a unique media type. Otherwise for Bacula versions 1.38 and older, your restores may not work because Bacula will assume that you can mount any Media Type with the same name on any Device associated with that Media Type. This is possible with tape drives, but with disk drives, unless you are very clever you cannot mount a Volume in any directory – this can be done by creating an appropriate soft link.

Currently Bacula permits only a single Media Type per Storage and Device definition. Consequently, if you have a drive that supports more than one Media Type, you can give a unique string to Volumes with different intrinsic Media Type (Media Type = DDS-3-4 for DDS-3 and DDS-4 types), but then those volumes will only be mounted on drives indicated with the dual type (DDS-3-4).

If you want to tie Bacula to using a single Storage daemon or drive, you must specify a unique Media Type for that drive. This is an important point that should be carefully understood. Note, this applies equally to Disk Volumes. If you define more than one disk Device resource in your Storage daemon's conf file, the Volumes on those two devices are in fact incompatible because one can not be mounted on the other device since they are found in different directories. For this reason, you probably should use two different Media Types for your two disk Devices (even though you might think of them as both being File types). You can find more on this subject in the Basic Volume Management chapter of this manual.

The **MediaType** specified in the Director's Storage resource, **must** correspond to the **Media Type** specified in the `Device` resource of the **Storage daemon** configuration file. This directive is required, and it is used by the Director and the Storage daemon to ensure that a Volume automatically selected from the Pool corresponds to the physical device. If a Storage daemon handles multiple devices (e.g. will write to various file Volumes on different partitions), this directive allows you to specify exactly which device.

As mentioned above, the value specified in the Director's Storage resource must agree with the value specified in the Device resource in the **Storage daemon's** configuration file. It is also an additional check so that you don't try to write data for a DLT onto an 8mm device.

Autochanger = <yes|no> If you specify yes for this command (the default is no), when you use the `label` command or the `add` command to create a new Volume, Bacula will also request the Autochanger Slot number. This simplifies creating database entries for Volumes in an autochanger. If you forget to specify the Slot, the autochanger will not be used. However, you may modify the Slot associated with a Volume at any time by using the `update`

volume or update slots command in the console program. When **Autochanger** is enabled, the algorithm used by Bacula to search for available volumes will be modified to consider only Volumes that are known to be in the autochanger's magazine. If no **in changer** volume is found, Bacula will attempt recycling, pruning, ..., and if still no volume is found, Bacula will search for any volume whether or not in the magazine. By privileging in changer volumes, this procedure minimizes operator intervention. The default is no.

For the autochanger to be used, you must also specify **Autochanger**=yes in the Device Resource in the Storage daemon's configuration file as well as other important Storage daemon configuration information. Please consult the Using Autochangers manual of this chapter for the details of using autochangers. You can modify any additional Storage resources that correspond to devices that are part of the Autochanger device. Instead of the previous **Autochanger**=yes directive, the configuration should be modified to be **Autochanger**=xxx where xxx is the name of the Autochanger.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs with the current Storage resource that can run concurrently. Note, this directive limits only Jobs for Jobs using this Storage daemon. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Client resources will also apply in addition to any limit specified here. The default is set to 1, but you may set it to a larger number. However, if you set the Storage daemon's number of concurrent jobs greater than one, we recommend that you read the waring documented under Maximum Concurrent Jobs in the Director's resource or simply turn data spooling on as documented in the Data Spooling chapter of this manual.

Maximum Concurrent Read Jobs = <number> The main purpose is to limit the number of concurrent Copy, Migration, and VirtualFull jobs so that they don't monopolize all the Storage drives causing a deadlock situation where all the drives are allocated for reading but none remain for writing. This deadlock situation can occur when running multiple simultaneous Copy, Migration, and VirtualFull jobs.

The default value is set to 0 (zero), which means there is no limit on the number of read jobs. Note, limiting the read jobs does not apply to Restore jobs, which are normally started by hand. A reasonable value for this directive is one half the number of drives that the Storage resource has rounded down. Doing so, will leave the same number of drives for writing and will generally avoid over committing drives and a deadlock.

AllowCompression = <yes|no> This directive is optional, and if you specify no (the default is yes), it will cause backups jobs running on this storage resource to run without client File Daemon compression. This effectively overrides compression options in FileSets used by jobs which use this storage resource.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Storage resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the setsockopt TCP_KEEPIDLE function. The default value is 300s.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[6] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

[6] https://en.wikipedia.org/wiki/X.509

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because

the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following is an example of a valid Storage resource definition:

```
# Definition of tape storage device
Storage {
  Name = DLTDrive
  Address = lpmatou
  Password = storage_password # password for Storage daemon
  Device = "HP DLT 80"    # same as Device in Storage daemon
  Media Type = DLT8000    # same as MediaType in Storage daemon
}
```

22.15 The Autochanger Resources

Each autochanger that you have defined in an Autochanger resource in the Storage daemon's bacula-sd.conf file, must have a corresponding Autochanger resource defined in the Director's bacula-dir.conf file. The Autochanger resource uses the same directives as the Storage resource defined 22.13 on page 275.

Normally you will already have a Storage resource that points to the Storage daemon's Autochanger resource. Thus you need only to change the name of the Storage resource to Autochanger. In addition the **Autochanger**=yes directive is not needed in the Director's Autochanger resource, since the resource name is Autochanger, the Director already knows that it represents an autochanger.

In addition to the above change (Storage to Autochanger), you must modify any additional Storage resources that correspond to devices that are part of the Autochanger device. Instead of the previous **Autochanger**=yes directive, the configuration should be modified to be **Autochanger**=xxx where xxx is the name of the Autochanger.

For example, in the bacula-dir.conf file:

```
Autochanger {             # New resource
  Name = Changer-1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO-Changer-1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 50
}

Storage {
  Name = Changer-1-Drive0
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive0
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}
```

```
Storage {
  Name = Changer-1-Drive1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

...
```

Note that Storage resources `Changer-1-Drive0` and `Changer-1-Drive1` are not required since they make up part of an autochanger, and normally, Jobs refer only to the Autochanger resource. However, by referring to those Storage definitions in a Job, you will use only the indicated drive. This is not normally what you want to do, but it is very useful and often used for reserving a drive for restores. See the Storage daemon example configuration below and the use of **AutoSelect**=no.

22.16 The Pool Resource

The Pool resource defines the set of storage Volumes (tapes or files) to be used by Bacula to write the data. By configuring different Pools, you can determine which set of Volumes (media) receives the backup data. This permits, for example, to store all full backup data on one set of Volumes and all incremental backups on another set of Volumes. Alternatively, you could assign a different set of Volumes to each machine that you backup. This is most easily done by defining multiple Pools.

Another important aspect of a Pool is that it contains the default attributes (Maximum Jobs, Retention Period, Recycle flag, ...) that will be given to a Volume when it is created. This avoids the need for you to answer a large number of questions when labeling a new Volume. Each of these attributes can later be changed on a Volume by Volume basis using the `update` command in the console program. Note that you must explicitly specify which Pool Bacula is to use with each Job. Bacula will not automatically search for the correct Pool.

Most often in Bacula installations all backups for all machines (Clients) go to a single set of Volumes. In this case, you will probably only use the **Default** Pool. If your backup strategy calls for you to mount a different tape each day, you will probably want to define a separate Pool for each day. For more information on this subject, please see the Backup Strategies chapter of this manual.

To use a Pool, there are three distinct steps. First the Pool must be defined in the Director's configuration file. Then the Pool must be written to the Catalog database. This is done automatically by the Director each time that it starts, or alternatively can be done using the `create` command in the console program. Finally, if you change the Pool definition in the Director's configuration file and restart Bacula, the pool will be updated alternatively you can use the `update pool` console command to refresh the database image. It is this database image rather than the Director's resource image that is used for the default Volume attributes. Note, for the pool to be automatically created or updated, it must be explicitly referenced by a Job resource.

Next the physical media must be labeled. The labeling can either be done with the `label` command in the `console` program or using the `btape` program. The preferred method is to use the `label` command in the `console` program.

Finally, you must add Volume names (and their attributes) to the Pool. For Volumes to be used by Bacula they must be of the same **Media Type** as the archive device specified for the job (i.e. if you are going to back up to a DLT device, the Pool must have DLT volumes defined since 8mm volumes cannot be mounted on a DLT drive). The **Media Type** has particular importance

if you are backing up to files. When running a Job, you must explicitly specify which Pool to use. Bacula will then automatically select the next Volume to use from the Pool, but it will ensure that the **Media Type** of any Volume selected from the Pool is identical to that required by the Storage resource you have specified for the Job.

If you use the `label` command in the console program to label the Volumes, they will automatically be added to the Pool, so this last step is not normally required.

It is also possible to add Volumes to the database without explicitly labeling the physical volume. This is done with the `add` console command.

As previously mentioned, each time Bacula starts, it scans all the Pools associated with each Catalog, and if the database record does not already exist, it will be created from the Pool Resource definition. Bacula probably should do an `update pool` if you change the Pool definition, but currently, you must do this manually using the `update pool` command in the Console program.

The Pool Resource defined in the Director's configuration file (`bacula-dir.conf`) may contain the following directives:

Pool Start of the Pool resource. There must be at least one Pool resource defined.

Name = <name> The name of the pool. For most applications, you will use the default pool name Default. This directive is required.

Maximum Volumes = <number> This directive specifies the maximum number of volumes (tapes or files) contained in the pool. This directive is optional, if omitted or set to zero, any number of volumes will be permitted. In general, this directive is useful for Autochangers where there is a fixed number of Volumes, or for File storage where you wish to ensure that the backups made to disk files do not become too numerous or consume too much space.

This directive is only respected in case of volumes automatically created by Bacula. If you add volumes to a pool manually with the `label` command, it is possible to have more volumes in a pool than specified by **Maximum Volumes**.

Maximum Pool Bytes = <size> This directive specifies the total maximum size of volumes (tapes or files) contained in the pool. This directive is optional, if omitted or set to zero, any size will be permitted. The current size of the pool can be displayed with `llist pool` command. All type of volumes with any status are counted in the pool size. To exclude *Pruned* and *Recycled* volumes from the quota, the **RecyclePool** directive can be used to send back pruned volumes to the Scratch pool for example.

Pool Type = <type> This directive defines the pool type, which corresponds to the type of Job being run. It is required and may be one of the following:

Backup

*Archive

*Cloned

*Migration

*Copy

*Save

Note, only Backup is current implemented.

Storage = <storage-resource-name> The Storage directive defines the name of the storage services where you want to backup the FileSet data. For additional details, see the Storage Resource Chapter of this manual. The Storage resource may also be specified in the Job resource, but the value, if any, in the Pool resource overrides any value in the Job. This Storage resource definition is not required by either the Job resource or in the Pool, but it must be specified in one or the other. If not configuration error will result.

Use Volume Once = <yes|no> This directive if set to yes specifies that each volume is to be used only once. This is most useful when the Media is a file and you want a new file for each backup that is done. The default is no (i.e. use volume any number of times). This directive will most likely be phased out (deprecated), so you are recommended to use **Maximum Volume Jobs**=1 instead.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the update command in the Console.

Please see the notes below under **Maximum Volume Jobs** concerning using this directive with multiple simultaneous jobs.

Maximum Volume Jobs = <positive-integer> This directive specifies the maximum number of Jobs that can be written to the Volume. If you specify zero (0) (the default), there is no limit. Otherwise, when the number of Jobs backed up to the Volume equals <positive-integer> the Volume will be marked *Used*. When the Volume is marked *Used* it can no longer be used for appending Jobs, much like the *Full* status. A Volume that is marked *Used* or *Full* can be recycled if recycling is enabled, and thus used again. By setting **MaximumVolumeJobs** to 1, you get the same effect as setting **UseVolumeOnce**=yes.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the update command in the Console.

If you are running multiple simultaneous jobs, this directive may not work correctly because when a drive is reserved for a job, this directive is not taken into account, so multiple jobs may try to start writing to the Volume. At some point, when the Media record is updated, multiple simultaneous jobs may fail since the Volume can no longer be written.

Maximum Volume Files = <positive-integer> This directive specifies the maximum number of files that can be written to the Volume. If you specify zero (0, the default), there is no limit. Otherwise, when the number of files written to the Volume equals <positive-integer> the Volume will be marked *Used*. When the Volume is marked *Used* it can no longer be used for appending Jobs, much like the *Full* status, but it can be recycled if recycling is enabled and thus used again. This value is checked and the *Used* status is set only at the end of a job that writes to the particular volume.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the update command in the Console.

Maximum Volume Bytes = <size> This directive specifies the maximum number of bytes that can be written to the Volume. If you specify zero (0,the default), there is no limit except the physical size of the Volume. Otherwise, when the number of bytes written to the Volume equals <size> the Volume will be marked *Full*. When the Volume is marked *Full* it can no longer be used for appending Jobs, but it can be recycled if recycling is enabled, and thus the Volume can be re-used after recycling. The size specified is checked just before each block is written to the Volume and if the Volume size would exceed the specified Maximum Volume Bytes the *Full* status will be set and the Job will request the next available Volume to continue.

This directive is particularly useful for restricting the size of disk volumes, and will work correctly even in the case of multiple simultaneous jobs writing to the volume.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the update command in the Console.

Volume Use Duration = <time-period-specification> The Volume Use Duration directive defines the time period that the Volume can be written beginning from the time of first data write to the Volume. If the time-period specified is zero (0,the default), the Volume can be

written indefinitely. Otherwise, the next time a job runs that wants to access this Volume, and the time period from the first write to the volume (the first Job written) exceeds the time-period-specification, the Volume will be marked *Used*, which means that no more Jobs can be appended to the Volume, but it may be recycled if recycling is enabled. Using the command `status dir` applies algorithms similar to running jobs, so during such a command, the Volume status may also be changed. Once the Volume is recycled, it will be available for use again.

You might use this directive, for example, if you have a Volume used for Incremental backups, and Volumes used for Weekly Full backups. Once the Full backup is done, you will want to use a different Incremental Volume. This can be accomplished by setting the Volume Use Duration for the Incremental Volume to six days. I.e. it will be used for the 6 days following a Full save, then a different Incremental volume will be used. Be careful about setting the duration to short periods such as 23 hours, or you might experience problems of Bacula waiting for a tape over the weekend only to complete the backups Monday morning when an operator mounts a new tape.

The use duration is checked and the *Used* status is set only at the end of a job that writes to the particular volume, which means that even though the use duration may have expired, the catalog entry will not be updated until the next job that uses this volume is run. This directive is not intended to be used to limit volume sizes and may not work as expected (i.e. will fail jobs) if the use duration expires while multiple simultaneous jobs are writing to the volume.

Please note that the value defined by this directive in the `bacula-dir.conf` file is the default value used when a Volume is created. Once the volume is created, changing the value in the `bacula-dir.conf` file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update volume** command (command 1.5 page 18) in the Bacula Community Edition Console manual.

Catalog Files = <yes|no> This directive defines whether or not you want the names of the files that were saved to be put into the catalog. The default is yes. The advantage of specifying **Catalog Files**=No is that you will have a significantly smaller Catalog database. The disadvantage is that you will not be able to produce a Catalog listing of the files backed up for each Job (this is often called Browsing). Also, without the File entries in the catalog, you will not be able to use the Console `restore` command nor any other command that references File entries.

AutoPrune = <yes|no> If AutoPrune is set to yes (default), Bacula (version 1.20 or greater) will automatically apply the Volume Retention period when new Volume is needed and no appendable Volumes exist in the Pool. Volume pruning causes expired Jobs (older than the **Volume Retention** period) to be deleted from the Catalog and permits possible recycling of the Volume.

Volume Retention = <time-period-specification> The **Volume Retention** directive defines the longest amount of time that Bacula will keep records associated with the Volume in the Catalog database after the end time of each Job written to the Volume. When this time period expires, and if **AutoPrune** is set to yes Bacula may prune (remove) Job records that are older than the specified Volume Retention period if it is necessary to free up a Volume. Note, it is also possible for all the Job and File records to be pruned before the Volume Retention period if Job and File Retention periods are configured to a lower value. In that case the Volume can then be marked Pruned and subsequently recycled prior to expiration of the Volume Retention period.

Recycling will not occur until it is absolutely necessary to free up a volume (i.e. no other writable volume exists). All File records associated with pruned Jobs are also pruned. The time may be specified as seconds, minutes, hours, days, weeks, months, quarters, or years. The **Volume Retention** is applied independently of the **Job Retention** and the **File Retention** periods defined in the Client resource. This means that all the retention periods are applied in turn and that the shorter period is the one that effectively takes precedence. Note, that when the **Volume Retention** period has been reached, and it is necessary to obtain a new volume, Bacula will prune both the Job and the File records. And the inverse is also true that if all the Job and File records that refer to a Volume

were already pruned, then the Volume may be recycled regardless of its retention period. Pruning may also occur during a `status dir` command because it uses similar algorithms for finding the next available Volume.

It is important to know that when the Volume Retention period expires, or all the Job and File records have been pruned that refer to a Volume, Bacula does not automatically recycle a Volume. It attempts to keep the Volume data intact as long as possible before over writing the Volume.

By defining multiple Pools with different Volume Retention periods, you may effectively have a set of tapes that is recycled weekly, another Pool of tapes that is recycled monthly and so on. However, one must keep in mind that if your **Volume Retention** period is too short, it may prune the last valid Full backup, and hence until the next Full backup is done, you will not have a complete backup of your system, and in addition, the next Incremental or Differential backup will be promoted to a Full backup. As a consequence, the minimum **Volume Retention** period should be at twice the interval of your Full backups. This means that if you do a Full backup once a month, the minimum Volume retention period should be two months.

The default Volume retention period is 365 days, and either the default or the value defined by this directive in the `bacula-dir.conf` file is the default value used when a Volume is created. Once the volume is created, changing the value in the `bacula-dir.conf` file will not change what is stored for the Volume. To change the value for an existing Volume you must use the `update` command in the Console.

To disable the **Volume Retention** feature, it is possible to set the directive to 0. When disabled, the pruning will be done only on the Job Retention directives and the "ExpiresIn" information available in the `list volume` output is not available.

Cache Retention = <time-period-specification> The **Cache Retention** directive defines the longest amount of time that bacula will keep Parts associated with the Volume in the Storage daemon Cache directory after a successful upload to the Cloud. When this time period expires, and if the `cloud prune` command is issued, Bacula may prune (remove) Parts that are older than the specified **Cache Retention** period.

Note, it is also possible for all the Parts to be removed before the Cache Retention period is reached. In that case the `cloud truncate` command must be used.

Action On Purge = <Truncate The directive **ActionOnPurge**=Truncate instructs Bacula to permit the Volume to be truncated after it has been purged. Note: the ActionOnPurge is a bit misleading since the volume is not actually truncated when it is purged, but is enabled to be truncated. The actual truncation is done with the `truncate` command.

To actually truncate a Volume, you must first set the ActionOnPurge to Truncate in the Pool, then you must ensure that any existing Volumes also have this information in them, by doing an `update Volumes` comand. Finally, after the Volume has been purged, you may then truncate it. It is useful to prevent disk based volumes from consuming too much space. See below for more details of how to ensure Volumes are truncated after being purged.

First set the Pool to permit truncation.

```
Pool {
  Name = Default
  Action On Purge = Truncate
  ...
}
```

Then assuming a Volume has been Purged, you can schedule truncate operation at the end of your CatalogBackup job like in this example:

```
Job {
 Name = CatalogBackup
 ...
 RunScript {
   RunsWhen=After
   RunsOnClient=No
   Console = "truncate Volume allpools storage=File"
 }
}
```

ScratchPool = <pool-resource-name> This directive permits specifing a specific scratch Pool to be used for the Job. This pool will replace the default scratch pool named *Scratch* for volume selection. For more information about scratch pools see Scratch Pool section of this manual. This directive is useful when using multiple storage devices that share the same MediaType or when you want to dedicate volumes to a particular set of pools.

RecyclePool = <pool-resource-name> This directive defines to which pool the Volume will be placed (moved) when it is recycled. Without this directive, a Volume will remain in the same pool when it is recycled. With this directive, it will be moved automatically to any existing pool during a recycle. This directive is probably most useful when defined in the Scratch pool, so that volumes will be recycled back into the Scratch pool. For more on the see the Scratch Pool section of this manual.

Although this directive is called RecyclePool, the Volume in question is actually moved from its current pool to the one you specify on this directive when Bacula prunes the Volume and discovers that there are no records left in the catalog and hence marks it as *Purged*.

Recycle = <yes|no> This directive specifies whether or not Purged Volumes may be recycled. If it is set to yes (default) and Bacula needs a volume but finds none that are appendable, it will search for and recycle (reuse) Purged Volumes (i.e. volumes with all the Jobs and Files expired and thus deleted from the Catalog). If the Volume is recycled, all previous data written to that Volume will be overwritten. If Recycle is set to no, the Volume will not be recycled, and hence, the data will remain valid. If you want to reuse (re-write) the Volume, and the recycle flag is no (0 in the catalog), you may manually set the recycle flag (`update` command) for a Volume to be reused.

Please note that the value defined by this directive in the `bacula-dir.conf` file is the default value used when a Volume is created. Once the volume is created, changing the value in the `bacula-dir.conf` file will not change what is stored for the Volume. To change the value for an existing Volume you must use the `update` command in the Console.

When all Job and File records have been pruned or purged from the catalog for a particular Volume, if that Volume is marked as *Full* or *Used*, it will then be marked as *Purged*. Only Volumes marked as *Purged* will be considered to be converted to the *Recycled* state if the **Recycle** directive is set to yes.

Recycle Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage daemon and none are available. The catalog is then **pruned** respecting the retention periods of all Files and Jobs written to this Volume. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and as such it is **much** better to use this directive than the Purge Oldest Volume.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and you have specified the correct retention periods.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and Bacula needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is no.

Recycle Current Volume = <yes|no> If Bacula needs a new Volume, this directive instructs Bacula to Prune the volume respecting the Job and File retention periods. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and thus it is **much** better to use it rather than the Purge Oldest Volume directive.

This directive can be useful if you have a fixed number of Volumes in the Pool, you want to cycle through them, and you have specified retention periods that prune Volumes before you have cycled through the Volume in the Pool.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and Bacula needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is no.

Purge Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage daemon and none are available. The catalog is then **purged** irrespective of retention periods of all Files and Jobs written to this Volume. The Volume is then recycled and will be used as the next Volume to be written. This directive overrides any Job, File, or Volume retention periods that you may have specified.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and reusing the oldest one when all Volumes are full, but you don't want to worry about setting proper retention periods. However, by using this option you risk losing valuable data.

Please be aware that Purge Oldest Volume disregards all retention periods. If you have only a single Volume defined and you turn this variable on, that Volume will always be immediately overwritten when it fills! So at a minimum, ensure that you have a decent number of Volumes in your Pool before running any jobs. If you want retention periods to apply do not use this directive. To specify a retention period, use the **Volume Retention** directive (see above).

We **highly** recommend against using this directive, because it is sure that some day, Bacula will recycle a Volume that contains current data. The default is no.

File Retention = <time-period-specification> The File Retention directive defines the length of time that Bacula will keep File records in the Catalog database after the End time of the Job corresponding to the File records.

This directive takes precedence over Client directives of the same name. For example, you can decide to increase Retention times for Archive or OffSite Pool.

Note, this affects only records in the catalog database. It does not affect your archive backups.

For more information see Client documentation about FileRetention

Job Retention = <time-period-specification> The Job Retention directive defines the length of time that Bacula will keep Job records in the Catalog database after the Job End time. As with the other retention periods, this affects only records in the catalog and not data in your archive backup.

This directive takes precedence over Client directives of the same name. For example, you can decide to increase Retention times for Archive or OffSite Pool.

For more information see Client side documentation JobRetention

Cleaning Prefix = <string> This directive defines a prefix string, which if it matches the beginning of a Volume name during labeling of a Volume, the Volume will be defined with the VolStatus set to *Cleaning* and thus Bacula will never attempt to use this tape. This is primarily for use with autochangers that accept barcodes where the convention is that barcodes beginning with CLN are treated as cleaning tapes.

Label Format = <format> This directive specifies the format of the labels contained in this pool. The format directive is used as a sort of template to create new Volume names during automatic Volume labeling.

The <format> should be specified in double quotes, and consists of letters, numbers and the special characters hyphen (-), underscore (_), colon (:), and period (.), which are the legal characters for a Volume name. The <format> should be enclosed in double quotes (").

In addition, the format may contain a number of variable expansion characters which will be expanded by a complex algorithm allowing you to create Volume names of many different formats. In all cases, the expansion process must resolve to the set of characters noted above that are legal Volume names. Generally, these variable expansion characters begin with a dollar sign (**$**) or a left bracket (**[**). If you specify variable expansion characters, you should always enclose the format with double quote characters (**"**). For more details on variable expansion, please see the **Variable Expansion** chapter (chapter 2 page 17) of the Bacula Community Edition Miscellaneous Guide.

If no variable expansion characters are found in the string, the Volume name will be formed from the <format> string appended with the a unique number that increases. If you do not remove volumes from the pool, this number should be the number of volumes plus one, but this is not guaranteed. The unique number will be edited as four digits with leading zeros. For example, with a **Label Format**="File-", the first volumes will be named File-0001, File-0002, ...

With the exception of Job specific variables, you can test your **LabelFormat** by using the **var** command (command 1.5 page 19) in the Bacula Community Edition Console manual.

```
Label Format="${Level}_${Type}_${Client}_${Year}-${Month:p/2/0/r}-${Day:p/2/0/r}"
```

Once defined, the name of the volume cannot be changed. When the volume is recycled, the volume can be used by an other Job at an other time, and possibly from an other Pool. In the example above, the volume defined with such name is probably not supposed to be recycled or reused.

In almost all cases, you should enclose the format specification (part after the equal sign) in double quotes.

In order for a Pool to be used during a Backup Job, the Pool must have at least one Volume associated with it. Volumes are created for a Pool using the label or the add commands in the Bacula bconsole program. In addition to adding Volumes to the Pool (i.e. putting the Volume names in the Catalog database), the physical Volume must be labeled with a valid Bacula software volume label before Bacula will accept the Volume. This will be automatically done if you use the label command. Bacula can automatically label Volumes if instructed to do so, but this feature is not yet fully implemented.

The following is an example of a valid Pool resource definition:

```
Pool {
  Name = Default
  Pool Type = Backup
}
```

22.16.1 The Scratch Pool

In general, you can give your Pools any name you wish, but there is one important restriction: the Pool named Scratch, if it exists behaves like a scratch pool of Volumes in that when Bacula needs a new Volume for writing and it cannot find one, it will look in the Scratch pool, and if it finds an available Volume, it will move it out of the Scratch pool into the Pool currently being used by the job.

22.17 The Catalog Resource

The Catalog Resource defines what catalog to use for the current job. Currently, Bacula can only handle a single database server (MySQL, PostgreSQL) that is defined when configuring Bacula. However, there may be as many Catalogs (databases) defined as you wish. For example, you may want each Client to have its own Catalog database, or you may want backup jobs to use one database and verify or restore jobs to use another database.

Since both MySQL and PostgreSQL are networked databases, they may reside either on the same machine as the Director or on a different machine on the network. See below for more details.

Catalog Start of the Catalog resource. At least one Catalog resource must be defined.

Name = <name> The name of the Catalog. No necessary relation to the database server name. This name will be specified in the Client resource directive indicating that all catalog data for that Client is maintained in this Catalog. This directive is required.

password = <password> This specifies the password to use when logging into the database. This directive is required.

DB Name = <name> This specifies the name of the database. If you use multiple catalogs (databases), you specify which one here. If you are using an external database server rather than the internal one, you must specify a name that is known to the server (i.e. you explicitly created the Bacula tables using this name. This directive is required.

user = <user> This specifies what user name to use to log into the database. This directive is required.

DB Socket = <socket-name> This is the name of a socket to use on the local host to connect to the database. This directive is used only by MySQL. Normally, if neither **DB Socket** or **DB Address** are specified, MySQL will use the default socket. If the DB Socket is specified, the MySQL server must reside on the same machine as the Director.

DB Address = <address> This is the host address of the database server. Normally, you would specify this instead of **DB Socket** if the database server is on another machine. In that case, you will also specify **DB Port**. This directive is used only by MySQL and PostgreSQL. This directive is optional.

DB Port = <port> This defines the port to be used in conjunction with **DB Address** to access the database if it is on another machine. This directive is used only by MySQL and PostgreSQL. This directive is optional.

The following is an example of a valid Catalog resource definition:

```
Catalog
{
  Name = MySQL
  dbname = bacula;
  user = bacula;
  password = ""                         # no password = no security
}
```

or for a Catalog on another machine:

```
Catalog
{
  Name = MySQL
  dbname = bacula
  user = bacula
  password = ""
  DB Address = remote.acme.com
  DB Port = 1234
}
```

22.18 The Messages Resource

For the details of the Messages Resource, please see the Messages Resource Chapter of this manual.

22.19 The Console Resource

As of Bacula version 1.33 and higher, there are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director's resource and consequently such consoles do not have a name as defined on a **Name**= directive. This is the kind of console that was initially implemented in versions prior to 1.33 and remains valid. Typically you would use it only for administrators.
- The second type of console, and new to version 1.33 and higher is a "named" console defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.

  This second type of console begins with absolutely no privileges except those explicitly specified in the Director's Console resource. Thus you can have multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges. As a default, these consoles can do absolutely nothing – no commands whatsoever. You give them privileges or rather access to commands and resources by specifying access control lists in the Director's Console resource. The ACLs are specified by a directive followed by a list of access names. Examples of this are shown below.
- The third type of console is similar to the above mentioned one in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the **Name**= directive, is the same as a Client name, that console is permitted to use the `SetIP` command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified. The following directives are permitted within the Director's configuration resource:

Name = <name> The name of the console. This name must match the name specified in the Console's configuration resource (much as is the case with Client definitions).

Password = <password> Specifies the password that must be supplied for a named Bacula Console to be authorized. The same password must appear in the `Console` resource of the Console configuration file. For added security, the password is never actually passed across the network but rather a challenge response hash code created with the password. This directive is required. If you have either `/dev/random` or `bc` on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process. However, it is preferable for security reasons to choose random text.

AuthenticationPlugin = <plugin-definition> Specifies the a plugin to use the authentication API framework which allows to configure a different set of authentication mechanisms (user credentials verification) using a dedicated Director plugins. It is called BPAM (Bacula Pluggable Authentication Modules). The first plugin available is a LDAP connector that is suitable to connect OpenLDAP or ActiveDirectory.

The new framework support standard `user/password` and `MFA` authentication schemes which are fully driven by external plugins. On the client side `bconsole` when noticied will perform user interaction to collect required credentials. Bacula will still support all previous authentication schemas including `CRAM-MD5` and `TLS`. You can even configure `TLS Authentication` together with new BPAM authentication raising required security level. BPAM authentication is available for named Console resources only.

To use this feature you have to load dedicated Director plugin from directory pointed by `Plugin Directory` Director resource configuration. This plugin should be listed in director status command. Then you should configure the Console resource to use this plugin for authentication with the `Authentication Plugin` directive together with a named Console configuration in `bconsole.conf` as described in Console Configuration chapter of this manual.

```
Console {
   Name = "ldapconsole"
   Password = "xxx"

   Authentication Plugin = "ldap:<parameters>"
}
```

where `parameters` are the space separated list of one or more plugin parameters:

url - LDAP Directory service location, i.e. "url=ldap://10.0.0.1/"

binddn - DN used to connect to LDAP Directory service to perform required `query`

bindpass - DN password used to connect to LDAP Directory service to perform required `query`

query - a query performed on LDAP Directory serice to find user for authentication. The `query` string is composed as `<basedn>/<filter>`. Where '`<basedn>` is a DN search starting point and `<filter>` is a standard LDAP search object filter which support dynamic string substitution: %u will be replaced by credential's username and %p by credential's password, i.e. `query=dc=bacula,dc=com/(cn=%u)`.

starttls - instruct BPAM LDAP Plugin to use **StartTLS** extension if LDAP Directory service will support it and follback to no TLS if this extension is not available.

starttlsforce - does the same what 'starttls' but resport error on fallback.

The working configuration examples:

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with OpenLDAP authentication example.

```
Console {
   Name = "bacula_ldap_console"
   Password = "xxx"

   # New directive (on a single line)
   Authentication Plugin = "ldap:url=ldap://ldapsrv/
      binddn=cn=root,dc=bacula,dc=com bindpass=secret
      query=dc=bacula,dc=com/(cn=%u) starttls"
   ...
}
```

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with Active Directory authentication example.

```
Console {
   Name = "bacula_ad_console"
   Password = "xxx"

   # New directive (on a single line)
 Authentication Plugin = "ldap:url=ldaps://ldapsrv/
  binddn=cn=bacula,ou=Users,dc=bacula,dc=com bindpass=secret
  query=dc=bacula,dc=com/(&(objectCategory=person)(objectClass=user)(sAMAccountName=%u))"
   ...
}
```

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[7] certificate. Any client certificate signed by a known-CA will be accepted.

[7] https://en.wikipedia.org/wiki/X.509

Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible

hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

JobACL = <name-list> This directive is used to specify a list of Job resource names that can be accessed by the console. Without this directive, the console cannot access any of the Director's Job resources. Multiple Job resource names may be specified by separating them with commas, and/or by specifying multiple JobACL directives. For example, the directive may be specified as:

```
JobACL = kernsave, "Backup client 1", "Backup client 2"
JobACL = "RestoreFiles"
```

With the above specification, the console can access the Director's resources for the four jobs named on the JobACL directives, but for no others.

ClientACL = <name-list> This directive is used to specify a list of `Client` resource names that can be accessed by the console.

RestoreClientACL = <name-list> This directive is used to specify a list of `Client` resource names that can be used by the console to restore files. The **ClientAcl** is not affected by the **RestoreClientACL** directive.

```
ClientAcl = localhost-fd         # backup and restore
RestoreClientAcl = test-fd       # restore only
BackupClientAcl = production-fd  # backup only
```

BackupClientACL = <name-list> This directive is used to specify a list of `Client` resource names that can be used by the console to backup files. The **ClientAcl** is not affected by the **RestoreClientACL** directive.

DirectoryACL = <name-list> This directive is used to specify a list of directories that can be accessed by a restore session. Without this directive, the console cannot restore any file. Multiple directories names may be specified by separating them with commas, and/or by specifying multiple **DirectoryACL** directives.

UserIdACL = <name-list> This directive is used to specify a list of `UID/GID` that can be accessed from a restore session. Without this directive, the console cannot restore any file. During the restore session, the Director will compute the restore list and will exclude files and directories that cannot be accessed. Bacula uses the LStat database field to retrieve `st_mode`, `st_uid` and `st_gid` information for each file and compare them with the UserIdACL elements. If a parent directory doesn't have a proper catalog entry, the access to this directory will be automatically granted.

`UID/GID` names are resolved with `getpwnam()` function within the Director. The User `UID/GID` mapping might be different from one system to an other.

Windows systems are not compatible with the **UserIdACL** feature. The use of **UserIdACL**=*all* is required to restore Windows systems from a restricted Console.

Multiple `UID/GID` names may be specified by separating them with commas, and/or by specifying multiple **UserIdACL** directives.

StorageACL = <name-list> This directive is used to specify a list of Storage resource names that can be accessed by the console.

ScheduleACL = <name-list> This directive is used to specify a list of Schedule resource names that can be accessed by the console.

PoolACL = <name-list> This directive is used to specify a list of Pool resource names that can be accessed by the console.

FileSetACL = <name-list> This directive is used to specify a list of FileSet resource names that can be accessed by the console.

CatalogACL = <name-list> This directive is used to specify a list of Catalog resource names that can be accessed by the console.

CommandACL = <name-list> This directive is used to specify a list of console commands that can be executed by the console.

WhereACL = <string> This directive permits you to specify where a restricted console can restore files. If this directive is not specified, only the default restore location is permitted (normally /tmp/bacula-restores). If *all* is specified any path the user enters will be accepted (not very secure), any other value specified (there may be multiple WhereACL directives) will restrict the user to use that path. For example, on a Unix system, if you specify "/", the file will be restored to the original location. This directive is untested.

Aside from Director resource names and console command names, the special keyword *all* can be specified in any of the above access control lists. When this keyword is present, any resource or command name (which ever is appropriate) will be accepted. For an example configuration file, please see the Console Configuration chapter of this manual.

Please note the *all* keyword must be used alone in an ACL list. The following definition will be interpreted as a "Glob Pattern" for any FileSet including the "all" string:

```
FileSetACL = fs-test*, *all*
```

or

```
FileSetACL = fs-test*
FileSetACL = *all*
```

All ACL Resource directives (**JobACL**, **ClientACL**, **FileSetACL**, ...) can accept "Glob Patterns" such as *.prod or 'job.client1.* to ease the configuration.

```
ClientAcl = test-*
JobACL = job-test-*
FileSetACL = fs-test*
```

22.20 The Counter Resource

The Counter Resource defines a counter variable that can be accessed by variable expansion used for creating Volume labels with the **LabelFormat** directive. See the LabelFormat directive in this chapter for more details.

Counter Start of the Counter resource. Counter directives are optional.

Name = <name> The name of the Counter. This is the name you will use in the variable expansion to reference the counter value.

Minimum = <integer> This specifies the minimum value that the counter can have. It also becomes the default. If not supplied, zero is assumed.

Maximum = <integer> This is the maximum value value that the counter can have. If not specified or set to zero, the counter can have a maximum value of 2,147,483,648 (2 to the 31 power). When the counter is incremented past this value, it is reset to the Minimum.

*WrapCounter = <counter-name> If this value is specified, when the counter is incremented past the maximum and thus reset to the minimum, the counter specified on the **WrapCounter** is incremented. (This is not currently implemented).

Catalog = <catalog-name> If this directive is specified, the counter and its values will be saved in the specified catalog. If this directive is not present, the counter will be redefined each time that Bacula is started.

22.21 The Statistics Resource

The `Statistics` Resource defines the statistic collector function that can send information to a Graphite instance, to a CSV file or to `bconsole` with the `statistics` command (See **??** on page ?? for more information).

Statistics Start of the `Statistics` resource. Statistics directives are optional.

Name = <name> The Statistics directive **name** is used by the system administrator. This directive is required.

Description = <string> The text field contains a description of the Statistics that will be displayed in the graphical user interface. This directive is optional.

Interval = <time-interval> The **Intervall** directive instructs the Statistics thread how long it should sleep between every collection iteration. This directive is optional and the default value is 300 seconds.

Type = <CSV|Graphite> The **Type** directive specifies the Statistics backend, which may be one of the following: CSV or Graphite. This directive is required.

CSV is a simple file level backend which saves all required metrics with the following format to the file: "<time>, <metric>, <value>\n"

Where <time> is a standard Unix time (a number of seconds from 1/01/1970) with local timezone as returned by a system call `time()`, <metric> is a Bacula metric string and <value> is a metric value which could be in numeric format (`int/float`) or a string True or False for boolean variable. The CSV backend requires the **File**= parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host**= and **Port**= directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and <metricspec> is a filter which enables to use * and ? characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with ! prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this statistics backend.

Example:

```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by statistics to distinguish between different installations or daemons. The prefix string will be added to metric name as: "<prefix>.<metric_name>" This directive is optional.

File = <filename> The File is used by the CSV statistics backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The statistics thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If statistics is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a mv shell command.

Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive **Type** is set to Graphite, the Host directive is required.

Host = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive **Type** is set to Graphite, the **Port** directive is required.

22.22 Example Director Configuration File

An example Director configuration file might be the following:

```
#
# Default Bacula Director Configuration file
#
#  The only thing that MUST be changed is to add one or more
#   file or directory names in the Include directive of the
#   FileSet resource.
#
#  For Bacula release 1.15 (5 March 2002) -- redhat
#
#  You might also want to change the default email address
#   from root to your address.  See the "mail" and "operator"
#   directives in the Messages resource.
#
Director {                            # define myself
  Name = rufus-dir
  QueryFile = "/home/kern/bacula/bin/query.sql"
  WorkingDirectory = "/home/kern/bacula/bin/working"
  PidDirectory = "/home/kern/bacula/bin/working"
  Password = "XkSfzu/Cf/wX4L8Zh4G4/yhCbpLcz3YVdmVoQvU3EyF/"
}
# Define the backup Job
Job {
  Name = "NightlySave"
  Type = Backup
  Level = Incremental                 # default
  Client=rufus-fd
  FileSet="Full Set"
  Schedule = "WeeklyCycle"
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
}
Job {
  Name = "Restore"
  Type = Restore
```

```
  Client=rufus-fd
  FileSet="Full Set"
  Where = /tmp/bacula-restores
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
}

# List of files to be backed up
FileSet {
  Name = "Full Set"
  Include {
    Options { signature=SHA1}
#
#  Put your list of files here, one per line or include an
#    external list with:
#
#    @file-name
#
#  Note: / backs up everything
  File = /
}
  Exclude {}
}
# When to do the backups
Schedule {
  Name = "WeeklyCycle"
  Run = level=Full sun at 2:05
  Run = level=Incremental mon-sat at 2:05
}
# Client (File Services) to backup
Client {
  Name = rufus-fd
  Address = rufus
  Catalog = MyCatalog
  Password = "MQk6lVinz4GG2hdIZk1dsKE/LxMZGo6znMHiD7t7vzF+"
  File Retention = 60d      # sixty day file retention
  Job Retention = 1y        # 1 year Job retention
  AutoPrune = yes           # Auto apply retention periods
}
# Definition of DLT tape storage device
Storage {
  Name = DLTDrive
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "HP DLT 80"      # same as Device in Storage daemon
  Media Type = DLT8000      # same as MediaType in Storage daemon
}
# Definition for a DLT autochanger device
Storage {
  Name = Autochanger
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "Autochanger"    # same as Device in Storage daemon
  Media Type = DLT-8000     # Different from DLTDrive
  Autochanger = yes
}
# Definition of DDS tape storage device
Storage {
  Name = SDT-10000
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = SDT-10000        # same as Device in Storage daemon
  Media Type = DDS-4        # same as MediaType in Storage daemon
}
# Definition of 8mm tape storage device
Storage {
  Name = "8mmDrive"
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "Exabyte 8mm"
  MediaType = "8mm"
}
# Definition of file storage device
Storage {
```

```
  Name = File
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = FileStorage
  Media Type = File
}
# Generic catalog service
Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}
# Reasonable message delivery -- send most everything to
#   the email address and to the console
Messages {
  Name = Standard
  mail = root@localhost = all, !skipped, !terminate
  operator = root@localhost = mount
  console = all, !skipped, !saved
}

# Default pool definition
Pool {
  Name = Default
  Pool Type = Backup
  AutoPrune = yes
  Recycle = yes
}
#
# Restricted console used by tray-monitor to get the status of the director
#
Console {
  Name = Monitor
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  CommandACL = status, .status
}
```

Chapter 23

Client/File daemon Configuration

The Client (or File Daemon) Configuration is one of the simpler ones to specify. Generally, other than changing the Client name so that error messages are easily identified, you will not need to modify the default Client configuration file.

For a general discussion of configuration file and resources including the data types recognized by Bacula, please see the Configuration chapter of this manual. The following Client Resource definitions must be defined:

- Client – to define what Clients are to be backed up.
- Director – to define the Director's name and its access password.
- Messages – to define where error and information messages are to be sent.

23.1 The Client Resource

The Client Resource (or FileDaemon) resource defines the name of the Client (as used by the Director) as well as the port on which the Client listens for Director connections.

Client (or FileDaemon) Start of the Client records. There must be one and only one Client resource in the configuration file, since it defines the properties of the current client program.

Name = <name> The client name that must be used by the Director when connecting. Generally, it is a good idea to use a name related to the machine so that error messages can be easily identified if you have multiple Clients. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the File daemon may put its status files. This directory should be used only by Bacula, but may be shared by other Bacula daemons provided the daemon names on the **Name** definition are unique for each daemon. This directive is required.

On Win32 systems, in some circumstances you may need to specify a drive letter in the specified working directory path. Also, please be sure that this directory is writable by the SYSTEM user otherwise restores may fail (the bootstrap file that is transferred to the File daemon from the Director is temporarily put in this directory before being passed to the Storage daemon).

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the File daemon may put its process Id file files. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. This record is required. Standard shell expansion of the <Directory> is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

Typically on Linux systems, you will set this to: /var/run. If you are not installing Bacula in the system directories, you can use the **Working Directory** as defined above.

Dedup Index Directory = <Directory> This directive is deprecated and replaced by "**Client Rehydration**".

This directive is optional and specifies a directory in which the File Daemon may put a index cache of all blocks read during a Backup job.

Standard shell expansion of the <Directory> is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

ClientRehydration = <Directory> This directive is optional and allows to try to do rehydration using existing local data on the Client at restore time. In some cases, the use of this directive permits to transfer less data over the network during a restore. The default value is no.

Heartbeat Interval = <time-interval> This record defines an interval of time in seconds. For each heartbeat that the File daemon receives from the Storage daemon, it will forward it to the Director. In addition, if no heartbeat has been received from the Storage daemon and thus forwarded the File daemon will send a heartbeat signal to the Director and to the Storage daemon to keep the channels active. The default interval is 300s. This feature is particularly useful if you have a router such as 3Com that does not follow Internet standards and times out a valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

If you continue getting broken pipe error messages despite using the Heartbeat Interval, and you are using Windows, you should consider upgrading your ethernet driver. This is a known problem with NVidia NForce 3 drivers (4.4.2 17/05/2004), or try the following workaround suggested by Thomas Simmons for Win32 machines:

Browse to: Start → Control Panel → Network Connections

Right click the connection for the nvidia adapter and select properties. Under the General tab, click "Configure...". Under the Advanced tab set "Checksum Offload" to disabled and click OK to save the change.

Lack of communications, or communications that get interrupted can also be caused by Linux firewalls where you have a rule that throttles connections or traffic.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs that should run concurrently. The default is set to 20, but you may set it to a larger number. Each contact from the Director (e.g. status request, job start request) is considered as a Job, so if you want to be able to do a `status` request in the console at the same time as a Job is running, you will need to set this value greater than 1. If set to a large value, please be careful to have this value higher than the **Maximum Concurrent Jobs** configured in the `Client` resource in the Director configuration file. Otherwise, backup jobs can fail due to the Director connection to FD be refused because Maximum Concurrent Jobs was exceeded on FD side.

Maximum Job Error Count = <number> where <number> is the error threshold for the Job, after reaching it Job will be failed. The default value is 1000. If this value is set to 0, job will continue to run no matter how many errors it encounters.

FDAddresses = <IP-address-specification> Specify the ports and addresses on which the File daemon listens for Director connections. Probably the simplest way to explain is to show an example:

```
FDAddresses  = {
   ip = { addr = 1.2.3.4; port = 1205; }
   ipv4 = {
       addr = 1.2.3.4; port = http; }
   ipv6 = {
       addr = 1.2.3.4;
       port = 1205;
   }
   ip = {
       addr = 1.2.3.4
```

```
        port = 1205
    }
    ip = { addr = 1.2.3.4 }
    ip = {
        addr = 201:220:222::2
    }
    ip = {
        addr = bluedot.thun.net
    }
}
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the /etc/services file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

FDPort = <port-number> This specifies the port number on which the Client listens for Director connections. It must agree with the FDPort specified in the Client resource of the Director's configuration file. The default is 9102.

FDAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Director connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the File daemon will bind to any available address (the default).

FDSourceAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Storage connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the kernel will choose the best address according to the routing table (the default).

SDConnectTimeout = <time-interval> This record defines an interval of time that the File daemon will try to connect to the Storage daemon. The default is 30 minutes. If no connection is made in the specified time interval, the File daemon cancels the Job.

Maximum Network Buffer Size = <bytes> where <bytes> specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is 65,536 bytes. The maximum value is 1,000,000 bytes.

Note, on certain Windows machines, there are reports that the transfer rates are very slow and this seems to be related to the default 65,536 size. On systems where the transfer rates seem abnormally slow compared to other systems, you might try setting the Maximum Network Buffer Size to 32,768 in both the File daemon and in the Storage daemon.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes per second that a job may use. You may specify the following speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfer with the value set with the Directive.

CommCompression = <yes|no> If the two Bacula components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled. In the case that the compression is not effective, Bacula turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports None in the Job report.

DisableCommand = <cmd> The **Disable Command** adds security to your File daemon by disabling certain commands globally. The commands that can be disabled are:

```
backup
cancel
setdebug=
setbandwidth=
estimate
fileset
JobId=
level =
restore
endrestore
session
status
.status
storage
verify
RunBeforeNow
RunBeforeJob
RunAfterJob
Run
accurate
```

One or more of these command keywords can be placed in quotes and separated by spaces on the **Disable Command** directive line. Note: the commands must be written exactly as they appear above.

SD Packet Check = <num-packets> The **SDPacketCheck** takes a positive integer. The integer if zero turns off the feature. If the integer is greater than zero, it is the number of packets that the FileDaemon will send to the Storage Daemon before to send a POLL request and wait for the Storage Daemon answer. The default value is 0. If the time between two POLL requests is too short (less than few seconds) and the number of bytes transfered is less than few hundred of MB, the value of the **SDPacketCheck** is increased dynamically.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (bacula-fd.conf), Director resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the Client resource, director configuration file. We can have the below client configuration in bacula-dir.conf:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[1] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in bacula-fd.conf, Director resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
```

[1] https://en.wikipedia.org/wiki/X.509

```
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

PKI Cipher This directive is optional and if specified will configure the data encryption to use a specific cipher. The default cipher is AES 128 CBC.

The following ciphers are available: aes128, aes192, aes256 and blowfish

PKI Digest This directive is optional and if specified will configure the data encryption to use a specific digest algorithm. The default cipher is SHA1 or SHA256 depending on the version of OpenSSL.

The following digest are available: md5, sha1, sha256.

PKI Encryption See the Data Encryption chapter of this manual.

PKI Signatures See the Data Encryption chapter of this manual.

PKI Keypair See the Data Encryption chapter of this manual.

PKI Master Key See the Data Encryption chapter of this manual.

The following is an example of a valid Client resource definition:

```
Client {                              # this is me
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  Pid Directory = $HOME/bacula/bin/working
}
```

23.2 The Director Resource

The Director resource defines the name and password of the Directors that are permitted to contact this Client.

Director Start of the Director records. There may be any number of Director resources in the Client configuration file. Each one specifies a Director that is allowed to connect to this Client.

Name = <name> The name of the Director that may contact this Client. This name must be the same as the name specified on the Director resource in the Director's configuration file. Note, the case (upper/lower) of the characters in the name are significant (i.e. S is not the same as s). This directive is required.

Password = <password> Specifies the password that must be supplied for a Director to be authorized. This password must be the same as the password specified in the Client resource in the Director's configuration file. This directive is required.

DirPort = <number> Specify the port to use to connect to the Director. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this directive is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director. This directive is required when **ConnectToDirector** is enabled.

ConnectToDirector = <yes|no> When the **ConnectToDirector** directive is set to `true`, the Client will contact the Director according to the `Schedule` rules. The connection initiated by the Client will be then used by the Director to start jobs or issue bconsole commands. If the **Schedule** directive is not set, the connection will be initiated when the file daemon starts. The connection will be reinitialized every **ReconnectionTime**. This directive can be useful if your File Daemon is behind a firewall that permits outgoing connections but not incoming connections.

ReconnectionTime = <time> When the Director resource of the FileDaemon is configured to connect the Director with the **ConnectToDirector** directive, the connection initiated by the FileDeamon to the Director will be reinitialized at a regular interval specified by the **ReconnectionTime** directive. The default value is 40 mins.

Schedule = <sched-resource> The **Schedule** directive defines what schedule is to be used for Client to connect the Director if the directive **ConnectToDirector** is set to `true`.

This directive is optional, and if left out, the Client will initiate a connection automatically at the start of the daemon. Although you may specify only a single Schedule resource for

any Director resource, the Schedule resource may contain multiple **Connect** directives, which allow you to initiate the Client connection at many different times, and each **Connect** directive permits to set the the **Max Connect Time** directive.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes per second that a job may use when started from this Director. You may specify the following speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

DisableCommand = <cmd> The **Disable Command** adds security to your File daemon by disabling certain commands for the current Director. More information about the syntax can be found on 23.1 on page 303.

Monitor = <yes|no> If Monitor is set to no (default), this director will have full access to this Client. If Monitor is set to yes, this director will only be able to fetch the current status of this Client.

Please note that if this director is being used by a Monitor, we highly recommend to set this directive to yes to avoid serious security problems.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (bacula-fd.conf), Director resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
```

```
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[2] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
```

[2] https://en.wikipedia.org/wiki/X.509

```
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

Thus multiple Directors may be authorized to use this Client's services. Each Director will have a different name, and normally a different password as well.

The following is an example of a valid Director resource definition:

```
#
# List Directors who are permitted to contact the File daemon
#
Director {
  Name = HeadMan
  Password = very_good                # password HeadMan must supply
}
Director {
  Name = Worker
  Password = not_as_good
  Monitor = Yes
}
```

23.3 The Schedule Resource

The Schedule resource provides a means of automatically scheduling the connection of the Client to the Director. Each Director resource can have a different Schedule resource.

Schedule Start of the Schedule directives. No `Schedule` resource is required. If no `Schedule` is used, the Client will connect automatically to the Director at the startup.

Name = <name> The name of the schedule being defined. The Name directive is required.

Enabled = <yes|no> This directive allows you to enable or disable the `Schedule` resource.

Connect = <Connect-overrides> <Date-time-specification> The Connect directive defines when a Client should connect to a Director. You may specify multiple **Connect** directives within a `Schedule` resource. If you do, they will all be applied (i.e. multiple schedules). If you have two **Connect** directives that start at the same time, two connections will start at the same time (well, within one second of each other). It is not recommended to have multiple connections at the same time.

Connect-options are specified as: **keyword=value** where the keyword is MaxConnectTime and the value is as defined on the respective directive formats for the Job resource. You may specify multiple **Connect-options** on one **Connect** directive by separating them with one or more spaces or by separating them with a trailing comma. For example:

MaxConnectTime=<time-spec> specifies how much time the connection will be attempt and active.

The <Date-time-specification> is similar to what exists for Job scheduling. See 22.5 on page 242 for more information.

An example schedule resource that is named WeeklyCycle and connects a director each Sunday at 2:05am and an on Monday through Saturday at 2:05am is:

```
Schedule {
  Name = "WeeklyCycle"
  Connect = MaxConnectTime=2h sun at 2:05
  Connect = MaxConnectTime=2h mon-sat at 2:05
}
```

In this example, the Job for this client should be schedule to start between 2:05 and 4:05 during the week. The Bacula administrator will be able to use commands such as `status client` or `estimate` betwen 2:05 and 4:05.

23.4 The Message Resource

Please see the Messages Resource Chapter of this manual for the details of the Messages Resource.

There must be at least one Message resource in the Client configuration file.

23.5 The Statistics Resource

The `Statistics` Resource defines the statistic collector function that can send information to a Graphite instance, to a CSV file or to `bconsole` with the `statistics` command (See **??** on page ?? for more information).

Statistics Start of the `Statistics` resource. Statistics directives are optional.

Name = <name> The Statistics directive **name** is used by the system administrator. This directive is required.

Description = <string> The text field contains a description of the `Statistics` resource that will be displayed in the graphical user interface. This directive is optional.

Interval = <time-interval> The **Intervall** directive instructs the Statistics collector thread how long it should sleep between every collection iteration. This directive is optional and the default value is 300 seconds.

Type = <CSV|Graphite> The **Type** directive specifies the Statistics backend, which may be one of the following: CSV or Graphite. This directive is required.

CSV is a simple file level backend which saves all required metrics with the following format to the file: "<time>, <metric>, <value>\n"

Where <time> is a standard Unix time (a number of seconds from 1/01/1970) with local timezone as returned by a system call `time()`, <metric> is a Bacula metric string and <value> is a metric value which could be in numeric format (`int/float`) or a string "True" or "False" for boolean variable. The CSV backend requires the **File=** parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host=** and **Port=** directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and <metricspec> is a filter which enables to use * and ? characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with ! prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this collector backend.

Example:

```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by collector to distinguish between different installations or daemons. The prefix string will be added to metric name as: "<prefix>.<metric_name>" This directive is optional.

File = <filename> The **File** is used by the CSV collector backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The collector thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If collector is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a mv shell command.

Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive Type is set to Graphite, the **Host** directive is required.

Host = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive Type is set to Graphite, the **Port** directive is required.

23.6 Example Client Configuration File

An example File Daemon configuration file might be the following:

```
#
# Default  Bacula File Daemon Configuration file
#
#  For Bacula release 1.35.2 (16 August 2004) -- gentoo 1.4.16
#
# There is not much to change here except perhaps to
#   set the Director's name and File daemon's name
#   to something more appropriate for your site.
#
#
# List Directors who are permitted to contact this File daemon
#
Director {
  Name = rufus-dir
  Password = "/LqPRkX++saVyQE7w7mmiFg/qxYc1kufww6FEyY/47jU"
}
#
# Restricted Director, used by tray-monitor to get the
#   status of the file daemon
#
Director {
  Name = rufus-mon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
  Monitor = yes
}
#
# "Global" File daemon configuration specifications
#
FileDaemon {                          # this is me
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  Pid Directory = $HOME/bacula/bin/working
}
# Send all messages except skipped files back to Director
Messages {
  Name = Standard
  director = rufus-dir = all, !skipped
}
```

Chapter 24

Storage Daemon Configuration

The Storage Daemon configuration file has relatively few resource definitions. However, due to the great variation in backup media and system capabilities, the storage daemon must be highly configurable. As a consequence, there are quite a large number of directives in the Device Resource definition that allow you to define all the characteristics of your Storage device (normally a tape drive). Fortunately, with modern storage devices, the defaults are sufficient, and very few directives are actually needed.

Examples of **Device** resource directives that are known to work for a number of common tape drives can be found in the `<bacula-src>/examples/devices` directory, and most will also be listed here.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual. The following Storage Resource definitions must be defined:

- Storage – to define the name of the Storage daemon.
- Director – to define the Director's name and his access password.
- Device – to define the characteristics of your storage device (tape drive).
- Messages – to define where error and information messages are to be sent.

24.1 Storage Resource

In general, the properties specified under the Storage resource define global properties of the Storage daemon. Each Storage daemon configuration file must have one and only one Storage resource definition.

Name = <Storage-Daemon-Name> Specifies the Name of the Storage daemon. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the Storage daemon may put its status files. This directory should be used only by **Bacula**, but may be shared by other Bacula daemons provided the names given to each daemon are unique. This directive is required

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the Storage daemon may put its process Id file files. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. This directive is required. Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

Typically on Linux systems, you will set this to: /var/run. If you are not installing Bacula in the system directories, you can use the **Working Directory** as defined above.

Dedup Directory = <Directory> This directive is mandatory when using Global Endpoint Deduplication feature, and specifies a directory in which the Storage daemon may put data block files for all devices of the type **dedup**. The **Dedup Directory** can be very large after some point, we advise you to use a logical volume manager to be able to extend the filesystem when needed with new disks.

Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

Dedup Index Directory = <Directory> This directive is optional and specifies a directory in which the Storage daemon may put its index files for all devices of the type **dedup**.

Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded. The default value is set to DedupDirectory when set.

Heartbeat Interval = <time-interval> This directive defines an interval of time in seconds. When the Storage daemon is waiting for the operator to mount a tape, each time interval, it will send a heartbeat signal to the File daemon. The default interval is 300s. This feature is particularly useful if you have a router such as 3Com that does not follow Internet standards and times out an valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

Client Connect Wait = <time-interval> This directive defines an interval of time in seconds that the Storage daemon will wait for a Client (the File daemon) to connect. The default is 30 minutes. Be aware that the longer the Storage daemon waits for a Client, the more resources will be tied up.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs that may run concurrently. The default is set to 20, but you may set it to a larger number. Each contact from the Director (e.g. status request, job start request) is considered as a Job, so if you want to be able to do a `status` request in the console at the same time as a Job is running, you will need to set this value greater than 1. To run simultaneous Jobs, you will need to set a number of other directives in the Director's configuration file. Which ones you set depend on what you want, but you will almost certainly need to set the **Maximum Concurrent Jobs** in the Storage resource in the Director's configuration file and possibly those in the Job and Client resources.

SDAddresses = <IP-address-specification> Specify the ports and addresses on which the Storage daemon will listen for Director connections. Normally, the default is sufficient and you do not need to specify this directive. Probably the simplest way to explain how this directive works is to show an example:

```
SDAddresses  = { ip = {
        addr = 1.2.3.4; port = 1205; }
    ipv4 = {
        addr = 1.2.3.4; port = http; }
    ipv6 = {
        addr = 1.2.3.4;
        port = 1205;
    }
    ip = {
        addr = 1.2.3.4
        port = 1205
    }
    ip = {
        addr = 1.2.3.4
    }
    ip = {
        addr = 201:220:222::2
    }
    ip = {
        addr = bluedot.thun.net
    }
}
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the `/etc/services` file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

Using this directive, you can replace both the SDPort and SDAddress directives shown below.

SDPort = <port-number> Specifies port number on which the Storage daemon listens for Director connections. The default is 9103.

SDAddress = <IP-Address> This directive is optional, and if it is specified, it will cause the Storage daemon server (for Director and File daemon connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this directive is not specified, the Storage daemon will bind to any available address (the default).

CommCompression = <yes|no> If the two Bacula components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled. In the case that the compression is not effective, Bacula turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports None in the Job report.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (bacula-fd.conf), Director resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the Client resource, director configuration file. We can have the below client configuration in bacula-dir.conf:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[1] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in bacula-fd.conf, Director resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
```

[1] https://en.wikipedia.org/wiki/X.509

```
    TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
    TLS Key = /opt/bacula/ssl/keys/client1_key.pem
  }
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following is a typical Storage daemon Storage definition.

```
#
# "Global" Storage daemon configuration specifications appear
# under the Storage resource.
#
Storage {
  Name = "Storage daemon"
  Address = localhost
  WorkingDirectory = "~/bacula/working"
  Pid    Directory = "~/bacula/working"
}
```

24.2 Director Resource

The Director resource specifies the Name of the Director which is permitted to use the services of the Storage daemon. There may be multiple Director resources. The Director Name and Password must match the corresponding values in the Director's configuration file.

Name = <Director-Name> Specifies the Name of the Director allowed to connect to the Storage daemon. This directive is required.

Password = <Director-password> Specifies the password that must be supplied by the above named Director. This directive is required.

Monitor = <yes|no> If Monitor is set to no (default), this director will have full access to this Storage daemon. If Monitor is set to yes, this director will only be able to fetch the current status of this Storage daemon.

Please note that if this director is being used by a Monitor, we highly recommend to set this directive to yes to avoid serious security problems.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
```

```
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[2] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
```

[2] https://en.wikipedia.org/wiki/X.509

```
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following is an example of a valid Director resource definition:

```
Director {
  Name = MainDirector
  Password = my_secret_password
}
```

24.3 Device Resource

The Device Resource specifies the details of each device (normally a tape drive) that can be used by the Storage daemon. There may be multiple Device resources for a single Storage daemon. In general, the properties specified within the Device resource are specific to the Device.

Name = <Device-Name> Specifies the Name that the Director will use when asking to backup or restore to or from to this device. This is the logical Device name, and may be any string up to 127 characters in length. It is generally a good idea to make it correspond to the English name of the backup device. The physical name of the device is specified on the **Archive Device** directive described below. The name you specify here is also used in your Director's conf file on the Device directive in its Storage resource.

Archive Device = <name-string> The specified **name-string** gives the system file name of the storage device managed by this storage daemon. This will usually be the device file name of a removable storage device (tape drive), for example `/dev/nst0` or `/dev/rmt/0mbn`. It may also be a directory name if you are archiving to disk storage. In this case, you must supply the full absolute path to the directory. When specifying a tape device, it is preferable that the "non-rewind" variant of the device file name be given. In addition, on systems such as Sun, which have multiple tape access methods, you must be sure to specify to use Berkeley I/O conventions with the device. The **b** in the Solaris (Sun) archive specification `/dev/rmt/0mbn` is what is needed in this case. Bacula does not support SysV tape drive behavior.

As noted above, normally the Archive Device is the name of a tape drive, but you may also specify an absolute path to an existing directory. If the Device is a directory Bacula will write to file storage in the specified directory, and the filename used will be the Volume name as specified in the Catalog. If you want to write into more than one directory (i.e. to spread the load to different disk drives), you will need to define two Device resources, each containing an Archive Device with a different directory. In addition to a tape device name or a directory name, Bacula will accept the name of a FIFO. A FIFO is a special kind of file that connects two programs via kernel memory. If a FIFO device is specified for a backup operation, you must have a program that reads what Bacula writes into the FIFO. When the Storage daemon starts the job, it will wait for **MaximumOpenWait** seconds for the read program to start reading, and then time it out and terminate the job. As a consequence, it is best to start the read program at the beginning of the job perhaps with the **RunBeforeJob** directive. For this kind of device, you never want to specify **AlwaysOpen**, because you want the Storage daemon to open it only when a job starts, so you must explicitly set it to no. Since a FIFO is a one way device, Bacula will not attempt to read a label of a FIFO device, but will simply write on it. To create a FIFO Volume in the catalog, use the `add` command rather than the `label` command to avoid attempting to write a label.

```
Device {
  Name = FifoStorage
  Media Type = Fifo
  Device Type = Fifo
  Archive Device = /tmp/fifo
  LabelMedia = yes
  Random Access = no
  AutomaticMount = no
  RemovableMedia = no
  MaximumOpenWait = 60
  AlwaysOpen = no
}
```

During a restore operation, if the Archive Device is a FIFO, Bacula will attempt to read from the FIFO, so you must have an external program that writes into the FIFO. Bacula will wait **MaximumOpenWait** seconds for the program to begin writing and will then time it out and terminate the job. As noted above, you may use the **RunBeforeJob** to start the writer program at the beginning of the job.

The Archive Device directive is required.

Device Type = <type-specification> The Device Type specification allows you to explicitly tell Bacula what kind of device you are defining. It the *type-specification* may be one of the following:

File Tells Bacula that the device is a file. It may either be a file defined on fixed medium or a removable filesystem such as USB. All files must be random access devices.

Tape The device is a tape device and thus is sequential access. Tape devices are controlled using ioctl() calls.

Fifo The device is a first-in-first out sequential access read-only or write-only device.

Aligned Tells Bacula that the device is special Aligned Device. Please see the specific Aligned User's guide for more information.

Dedup The device is a Deduplication device. The Storage Daemon will use the deduplication engine. Please see the specific Deduplication User's guide for more information.

Cloud The device is a Cloud device. The Storage Daemon will use the Cloud to upload/download volumes. Please see the specific Cloud User's guide for more information.

The Device Type directive is not required, and if not specified, Bacula will attempt to guess what kind of device has been specified using the Archive Device specification supplied. There are several advantages to explicitly specifying the Device Type. First, on some systems, block and character devices have the same type. Secondly, if you explicitly specify the Device Type, the mount point need not be defined until the device is opened. This is the case with most removable devices such as USB that are mounted by the HAL daemon. If the Device Type is not explicitly specified, then the mount point must exist when the Storage daemon starts.

Enabled = <yes|no> This directive allows you to enable or disable the `Device` resource.

Media Type = <name-string> The specified **name-string** names the type of media supported by this device, for example, "DLT7000". Media type names are arbitrary in that you set them to anything you want, but they must be known to the volume database to keep track of which storage daemons can read which volumes. In general, each different storage type should have a unique Media Type associated with it. The same **name-string** must appear in the appropriate Storage resource definition in the Director's configuration file.

Even though the names you assign are arbitrary (i.e. you choose the name you want), you should take care in specifying them because the Media Type is used to determine which storage device Bacula will select during restore. Thus you should probably use the same Media Type specification for all drives where the Media can be freely interchanged. This is not generally an issue if you have a single Storage daemon, but it is with multiple Storage daemons, especially if they have incompatible media.

For example, if you specify a Media Type of "DDS-4" then during the restore, Bacula will be able to choose any Storage Daemon that handles "DDS-4". If you have an autochanger, you might want to name the Media Type in a way that is unique to the autochanger, unless you wish to possibly use the Volumes in other drives. You should also ensure to have unique Media Type names if the Media is not compatible between drives. This specification is required for all devices.

In addition, if you are using disk storage, each Device resource will generally have a different mount point or directory. In order for Bacula to select the correct Device resource, each one must have a unique Media Type.

Autochanger = <yes|no> If yes, this device belongs to an automatic tape changer, and you must specify an **Autochanger** resource that points to the **Device** resources. You must also specify a **Changer Device**. If the Autochanger directive is set to no (default), the volume must be manually changed. You should also have an identical directive to the Storage resource in the Director's configuration file so that when labeling tapes you are prompted for the slot.

Changer Device = <name-string> The specified **name-string** must be the **generic SCSI** device name of the autochanger that corresponds to the normal read/write **Archive Device** specified in the Device resource. This generic SCSI device name should be specified if you have an autochanger or if you have a standard tape drive and want to use the **Alert Command** (see below). For example, on Linux systems, for an Archive Device name of `/dev/nst0`, you would specify `/dev/sg0` for the Changer Device name. Depending on your exact configuration, and the number of autochangers or the type of autochanger, what you specify here can vary. This directive is optional. See the Using Autochangers chapter of this manual for more details of using this and the following autochanger directives.

Changer Command = <name-string> The **name-string** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Normally, this directive will be specified only in the **AutoChanger** resource, which is then used for all devices. However, you may also specify the different **Changer Command** in each Device resource. Most frequently, you will specify the Bacula supplied `mtx-changer` script as follows:

```
Changer Command = "/path/mtx-changer %c %o %S %a %d"
```

and you will install the `mtx` on your system (found in the **depkgs** release). An example of this command is in the default `bacula-sd.conf` file. For more details on the substitution characters that may be specified to configure your autochanger please see the Autochangers chapter of this manual. For FreeBSD users, you might want to see one of the several `chio` scripts in `examples/autochangers`.

Control Device = <name-string> The control device is the SCSI control device that corresponds to the **Archive Device**. The correspondance can be done via the `lssci -g` command.

```
/opt/bacula# lsscsi -g
[1:0:0:0] tape    HP       Ultrium 4-SCSI   H61W  /dev/st0  /dev/sg0
[1:0:0:1] tape    HP       Ultrium 4-SCSI   H61W  /dev/st1  /dev/sg1
[1:0:0:2] mediumx HP       MSL G3 Series    E.00  -         /dev/sg2
```

Alert Command = <name-string> The **name-string** specifies an external program to be called at the completion of each Job after the device is released. The purpose of this command is to check for Tape Alerts, which are present when something is wrong with your tape drive (at least for most modern tape drives). The same substitution characters that may be specified in the Changer Command may also be used in this string. For more information, please see the Autochangers chapter of this manual.

The directive in the Device resource can call the `tapealert` script that is installed in the scripts directory. It is defined as follows:

```
Device {
    Name = ...
    Archive Device = /dev/nst0
    Alert Command = "/opt/bacula/scripts/tapealert %l"
    Control Device = /dev/sg1 # must be SCSI ctl for /dev/nst0
    ...
}
```

Once the above mentioned two directives (**Alert Command** and **Control Device**) are in place in each of your `Device` resources, Bacula will check for tape alerts at two points:

- After the Drive is used and it becomes idle.
- After each read or write error on the drive.

At each of the above times, Bacula will call the new `tapealert` script, which uses the `tapeinfo` program. The `tapeinfo` utility is part of the `apt` `sg3-utils` and `rpm` `sg3_utils` packages. Then for each tape alert that Bacula finds for that drive, it will emit a Job message that is either `INFO`, `WARNING`, or `FATAL` depending on the designation in the Tape Alert published by the T10 Technical Committee on SCSI Storage Interfaces[3]. For the specification, please see: `http://www.t10.org/ftp/t10/document.02/02-142r0.pdf`

Worm Command = <name-string> The name-string specifies an external program to be called when loading a new volume. The purpose of this command is to check if the current tape is a WORM[4] tape. The same substitution characters that may be specified in the **Changer Command** ay also be used in this string.

The directive in the `Device` resource can call the `isworm` script that is installed in the scripts directory. It is defined as follows:

[3] http://www.t10.org
[4] Write Once Read Many

```
Device {
  Name = ...
  Archive Device = /dev/nst0
  Worm Command = "/opt/bacula/scripts/isworm %l"
  Control Device = /dev/sg1 # must be SCSI ctl for /dev/nst0
  ...
}
```

Bacula will call the isworm script, which uses the tapeinfo and sdparm program.

Drive Index = <number> The **Drive Index** that you specify is passed to the mtx-changer script and is thus passed to the mtx program. By default, the Drive Index is zero, so if you have only one drive in your autochanger, everything will work normally. However, if you have multiple drives, you must specify multiple Bacula Device resources (one for each drive). The first Device should have the Drive Index set to 0, and the second Device Resource should contain a Drive Index set to 1, and so on. This will then permit you to use two or more drives in your autochanger. As of Bacula version 1.38.0, using the **Autochanger** resource, Bacula will automatically ensure that only one drive at a time uses the autochanger script, so you no longer need locking scripts as in the past – the default mtx-changer script works for any number of drives.

Autoselect = <yes|no> If this directive is set to yes (default), and the Device belongs to an autochanger, then when the Autochanger is referenced by the Director, this device can automatically be selected. If this directive is set to no, then the Device can only be referenced by directly using the Device name in the Director. This is useful for reserving a drive for something special such as a high priority backup or restore operations.

Maximum Concurrent Jobs = <num> **Maximum Concurrent Jobs** is a directive that permits setting the maximum number of Jobs that can run concurrently on a specified Device. Using this directive, it is possible to have different Jobs using multiple drives, because when the Maximum Concurrent Jobs limit is reached, the Storage Daemon will start new Jobs on any other available compatible drive. This facilitates writing to multiple drives with multiple Jobs that all use the same Pool.

Maximum Changer Wait = <time> This directive specifies the maximum time in seconds for Bacula to wait for an autochanger to change the volume. If this time is exceeded, Bacula will invalidate the Volume slot number stored in the catalog and try again. If no additional changer volumes exist, Bacula will ask the operator to intervene. The default is 5 minutes.

Maximum Rewind Wait = <time> This directive specifies the maximum time in seconds for Bacula to wait for a rewind before timing out. If this time is exceeded, Bacula will cancel the job. The default is 5 minutes.

Maximum Open Wait = <time> This directive specifies the maximum time in seconds that Bacula will wait for a device that is busy. The default is 5 minutes. If the device cannot be obtained, the current Job will be terminated in error. Bacula will re-attempt to open the drive the next time a Job starts that needs the the drive.

Always Open = <yes|no> If yes (default), Bacula will always keep the device open unless specifically **unmounted** by the Console program. This permits Bacula to ensure that the tape drive is always available, and properly positioned. If you set **AlwaysOpen** to no, **Bacula** will only open the drive when necessary, and at the end of the Job if no other Jobs are using the drive, it will be freed. The next time Bacula wants to append to a tape on a drive that was freed, Bacula will rewind the tape and position it to the end. To avoid unnecessary tape positioning and to minimize unnecessary operator intervention, it is highly recommended that **Always Open = yes**. This also ensures that the drive is available when Bacula needs it.

If you have **Always Open = yes** (recommended) and you want to use the drive for something else, simply use the unmount command in the Console program to release the drive. However, don't forget to remount the drive with mount when the drive is available or the next Bacula job will block.

For File storage, this directive is ignored. For a FIFO storage device, you must set this to no.

Please note that if you set this directive to no Bacula will release the tape drive between each job, and thus the next job will rewind the tape and position it to the end of the data. This can be a very time consuming operation. In addition, with this directive set to no, certain multiple drive autochanger operations will fail. We strongly recommend to keep **Always Open** set to yes

Volume Poll Interval = <time> If the time specified on this directive is non-zero, after asking the operator to mount a new volume Bacula will periodically poll (or read) the drive at the specified interval to see if a new volume has been mounted. If the time interval is zero, no polling will occur. The default value is 5 mins. This directive can be useful if you want to avoid operator intervention via the console. Instead, the operator can simply remove the old volume and insert the requested one, and Bacula on the next poll will recognize the new tape and continue. Please be aware that if you set this interval too small, you may excessively wear your tape drive if the old tape remains in the drive, since Bacula will read it on each poll. This can be avoided by ejecting the tape using the **Offline On Unmount** and the **Close on Poll** directives. However, if you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and Bacula will not be able to properly open the drive and may fail the job. For more information on this problem, please see the **description of Offline On Unmount** subsection (subsection 3.1 page 26) in the **Tape Testing** chapter (chapter 3 page 25) of the Bacula Community Edition Problems Resolution Guide.

Close on Poll = <yes|no> If yes, Bacula close the device (equivalent to an unmount except no mount is required) and reopen it at each poll. Normally this is not too useful unless you have the **Offline on Unmount** directive set, in which case the drive will be taken offline preventing wear on the tape during any future polling. Once the operator inserts a new tape, Bacula will recognize the drive on the next poll and automatically continue with the backup. Please see above more more details.

Sync On Close = <yes|no> If yes, Bacula will sync the device at the end of each job and when closing the device. Normally it might be useful if you store the data on network filesystems. The default value is no.

Maximum Open Wait = <time> This directive specifies the maximum amount of time in seconds that Bacula will wait for a device that is busy. The default is 5 minutes. If the device cannot be obtained, the current Job will be terminated in error. Bacula will re-attempt to open the drive the next time a Job starts that needs the the drive.

Removable Media = <yes|no> If yes, this device supports removable media (for example, tapes or CDs). If no, media cannot be removed (for example, an intermediate backup area on a hard disk). If **Removable media** is enabled on a File device (as opposed to a tape) the Storage daemon will assume that device may be something like a USB device that can be removed or a simply a removable harddisk. When attempting to open such a device, if the Volume is not found (for File devices, the Volume name is the same as the Filename), then the Storage daemon will search the entire device looking for likely Volume names, and for each one found, it will ask the Director if the Volume can be used. If so, the Storage daemon will use the first such Volume found. Thus it acts somewhat like a tape drive – if the correct Volume is not found, it looks at what actually is found, and if it is an appendable Volume, it will use it.

If the removable medium is not automatically mounted (e.g. udev), then you might consider using additional Storage daemon device directives such as **Requires Mount**, **Mount Point**, **Mount Command**, and **Unmount Command**, all of which can be used in conjunction with **Removable Media**.

Random Access = <yes|no> If yes, the archive device is assumed to be a random access medium which supports the **lseek** (or **lseek64** if Largefile is enabled during configuration) facility. This should be set to yes for all file systems such as USB, and fixed files. It should be set to no for non-random access devices such as tapes and named pipes.

Requires Mount = <yes|no> When this directive is enabled, the Storage daemon will submit a **Mount Command** before attempting to open the device. You must set this directive to yes for removable file systems such as USB devices that are not automatically mounted

by the operating system when plugged in or opened by Bacula. It should be set to no for all other devices such as tapes and fixed filesystems. It should also be set to no for any removable device that is automatically mounted by the operating system when opened (e.g. USB devices mounted by udev or hotplug). This directive indicates if the device requires to be mounted using the **Mount Command**. To be able to write devices need a mount, the following directives must also be defined: **Mount Point**, **Mount Command**, and **Unmount Command**.

Mount Point = <directory> Directory where the device can be mounted. This directive is used only for devices that have **Requires Mount** enabled such as USB file devices.

Mount Command = <name-string> This directive specifies the command that must be executed to mount devices such as many USB devices. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

See the Edit Codes section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Unmount Command = <name-string> This directive specifies the command that must be executed to unmount devices such as many USB devices. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount %m"
```

See the Edit Codes section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Block Checksum = <yes|no> You may turn off the Block Checksum (CRC32) code that Bacula uses when writing blocks to a Volume. Doing so can reduce the Storage daemon CPU usage slightly. It will also permit Bacula to read a Volume that has corrupted data.

The default is yes – i.e. the checksum is computed on write and checked on read.

We do not recommend to turn this off particularly on older tape drives or for disk Volumes where doing so may allow corrupted data to go undetected.

Minimum block size = <size-in-bytes> On most modern tape drives, you will not need or want to specify this directive, and if you do so, it will be to make Bacula use fixed block sizes. This statement applies only to non-random access devices (e.g. tape drives). Blocks written by the storage daemon to a non-random archive device will never be smaller than the given **size-in-bytes**. The Storage daemon will attempt to efficiently fill blocks with data received from active sessions but will, if necessary, add padding to a block to achieve the required minimum size.

To force the block size to be fixed, as is the case for some non-random access devices (tape drives), set the **Minimum block size** and the **Maximum block size** to the same value (zero included). The default is that both the minimum and maximum block size are zero and the default block size is 64,512 bytes.

For example, suppose you want a fixed block size of 100K bytes, then you would specify:

```
Minimum block size = 100K
Maximum block size = 100K
```

Please note that if you specify a fixed block size as shown above, the tape drive must either be in variable block size mode, or if it is in fixed block size mode, the block size (generally defined by mt) **must** be identical to the size specified in Bacula – otherwise when you attempt to re-read your Volumes, you will get an error.

If you want the block size to be variable but with a 64K minimum and 200K maximum (and default as well), you would specify:

```
Minimum block size = 64K
Maximum blocksize = 256K
```

Maximum block size = <size-in-bytes> On most modern tape drives, you will not need to specify this directive. If you do so, it will most likely be to reduce shoe-shine and improve performance on more modern LTO drives. The Storage daemon will always attempt to write blocks of the specified **size-in-bytes** to the archive device. As a consequence, this statement specifies both the default block size and the maximum block size. The size written never exceeds the given **size-in-bytes**. If adding data to a block would cause it to exceed the given maximum size, the block will be written to the archive device, and the new data will begin a new block.

If no value is specified or zero is specified, the Storage daemon will use a default block size of 64,512 bytes (126 * 512).

The maximum **size-in-bytes** possible is 4,000,000.

Hardware End of Medium = <yes|no> If no, the archive device is not required to support end of medium ioctl request, and the storage daemon will use the forward space file function to find the end of the recorded data. If yes, the archive device must support the `ioctl MTEOM` call, which will position the tape to the end of the recorded data. In addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space to the end of the recorded data, but they do not keep track of the file number. On Linux machines, the SCSI driver has a **fast-eod** option, which if set will cause the driver to lose track of the file number. You should ensure that this option is always turned off using the `mt` program.

Default setting for Hardware End of Medium is yes. This function is used before appending to a tape to ensure that no previously written data is lost. We recommend if you have a non-standard or unusual tape drive that you use the `btape` program to test your drive to see whether or not it supports this function. All modern (after 1998) tape drives support this feature.

Fast Forward Space File = <yes|no> If no, the archive device is not required to support keeping track of the file number (**MTIOCGET** ioctl) during forward space file. If yes, the archive device must support the `ioctl MTFSF` call, which virtually all drivers support, but in addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space, but they do not keep track of the file number or more seriously, they do not report end of medium.

Default setting for Fast Forward Space File is yes.

Use MTIOCGET = <yes|no> If no, the operating system is not required to support keeping track of the file number and reporting it in the (**MTIOCGET** ioctl). The default is yes. If you must set this to No, Bacula will do the proper file position determination, but it is very unfortunate because it means that tape movement is very inefficient. Fortunately, this operation system deficiency seems to be the case only on a few *BSD systems. Operating systems known to work correctly are Solaris, Linux and FreeBSD.

BSF at EOM = <yes|no> If no, the default, no special action is taken by Bacula with the End of Medium (end of tape) is reached because the tape will be positioned after the last EOF tape mark, and Bacula can append to the tape as desired. However, on some systems, such as FreeBSD, when Bacula reads the End of Medium (end of tape), the tape will be positioned after the second EOF tape mark (two successive EOF marks indicated End of Medium). If Bacula appends from that point, all the appended data will be lost. The solution for such systems is to specify **BSF at EOM** which causes Bacula to backspace over the second EOF mark. Determination of whether or not you need this directive is done using the `test` command in the `btape` program.

TWO EOF = <yes|no> If yes, Bacula will write two end of file marks when terminating a tape – i.e. after the last job or at the end of the medium. If no, the default, Bacula will only write one end of file to terminate the tape.

Backward Space Record = <yes|no> If yes, the archive device supports the `MTBSR ioctl` to backspace records. If no, this call is not used and the device must be rewound and advanced forward to the desired position. Default is yes for non random-access devices.

This function if enabled is used at the end of a Volume after writing the end of file and any ANSI/IBM labels to determine whether or not the last block was written correctly. If you turn this function off, the test will not be done. This causes no harm as the re-read process is precautionary rather than required.

Backward Space File = <yes|no> If yes, the archive device supports the **MTBSF** and **MTBSF ioctl**s to backspace over an end of file mark and to the start of a file. If no, these calls are not used and the device must be rewound and advanced forward to the desired position. Default is yes for non random-access devices.

Forward Space Record = <yes|no> If yes, the archive device must support the **MTFSR ioctl** to forward space over records. If no, data must be read in order to advance the position on the device. Default is yes for non random-access devices.

Forward Space File = <yes|no> If yes, the archive device must support the `MTFSF ioctl` to forward space by file marks. If no, data must be read to advance the position on the device. Default is yes for non random-access devices.

Offline On Unmount = <yes|no> The default for this directive is no. If yes the archive device must support the `MTOFFL ioctl` to rewind and take the volume offline. In this case, Bacula will issue the offline (eject) request before closing the device during the `unmount` command. If no Bacula will not attempt to offline the device before unmounting it. After an offline is issued, the cassette will be ejected thus **requiring operator intervention** to continue, and on some systems require an explicit load command to be issued (`mt -f /dev/xxx load`) before the system will recognize the tape. If you are using an autochanger, some devices require an offline to be issued prior to changing the volume. However, most devices do not and may get very confused.

If you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and Bacula will not be able to properly open the drive and may fail the job. For more information on this problem, please see the **description of Offline On Unmount** subsection (subsection 3.1 page 26) in the **Tape Testing** chapter (chapter 3 page 25) of the Bacula Community Edition Problems Resolution Guide.

Maximum Volume Size = <size> No more than **size** bytes will be written onto a given volume on the archive device. This directive is used mainly in testing Bacula to simulate a small Volume. It can also be useful if you wish to limit the size of a File Volume to say less than 2GB of data. In some rare cases of really antiquated tape drives that do not properly indicate when the end of a tape is reached during writing (though I have read about such drives, I have never personally encountered one). Please note, this directive is deprecated (being phased out) in favor of the **Maximum Volume Bytes** defined in the Director's configuration file.

Maximum File Size = <size> No more than **size** bytes will be written into a given logical file on the volume. Once this size is reached, an end of file mark is written on the volume and subsequent data are written into the next file. Breaking long sequences of data blocks with file marks permits quicker positioning to the start of a given stream of data and can improve recovery from read errors on the volume. The default is one Gigabyte. This directive creates EOF marks only on tape media. However, regardless of the medium type (tape, disk, USB ...) each time a the Maximum File Size is exceeded, a record is put into the catalog database that permits seeking to that position on the medium for restore operations. If you set this to a small value (e.g. 1MB), you will generate lots of database records (JobMedia) and may significantly increase CPU/disk overhead.

If you are configuring an LTO-3 or LTO-4 tape, you probably will want to set the **Maximum File Size** to 2GB to avoid making the drive stop to write an EOF mark.

Note, this directive does not limit the size of Volumes that Bacula will create regardless of whether they are tape or disk volumes. It changes only the number of EOF marks on a tape and the number of block positioning records (see below) that are generated. If you want to limit the size of all Volumes for a particular device, use the **Maximum Volume Size** directive (above), or use the **Maximum Volume Bytes** directive in the Director's Pool resource, which does the same thing but on a Pool (Volume) basis.

Maximum File Index = <size> Some data might include information about the actual position of a block in the data stream. This information is stored in the catalog inside the `FileMedia` table. By default, one index record will be created every 100MB of data. The index permits quicker positioning to the start of a given block in the Bacula Volume and can improve the Single Item Restore feature. If you set this to a small value (e.g. 1MB), you will generate lots of database records (FileMedia) and may significantly increase **CPU!**/disk overhead.

Maximum Part Size = <size> This directive allows one to specify the maximum size for each part. Smaller part sizes will reduce restore costs, but may require a small additional overhead to handle multiple parts. The maximum number of parts permitted in a Cloud Volume is 524,288. The maximum size of any given part is approximately 17.5TB.

Block Positioning = <yes|no> This directive tells Bacula not to use block positioning when doing restores. Turning this directive off can cause Bacula to be **extremely** slow when restoring files. You might use this directive if you wrote your tapes with Bacula in variable block mode (the default), but your drive was in fixed block mode. The default is yes.

Maximum Network Buffer Size = <bytes> where *bytes* specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is 32,768 bytes. The maximum value is 1,000,000 bytes.

The default size was chosen to be relatively large but not too big in the case that you are transmitting data over Internet. It is clear that on a high speed local network, you can increase this number and improve performance. For example, some users have found that if you use a value of 65,536 bytes they get five to ten times the throughput. Larger values for most users don't seem to improve performance. If you are interested in improving your backup speeds, this is definitely a place to experiment. You will probably also want to make the corresponding change in each of your File daemons conf files.

Maximum Spool Size = <bytes> where the bytes specify the maximum spool size for all jobs that are running. The default is no limit.

Maximum Job Spool Size = <bytes> where the bytes specify the maximum spool size for any one job that is running. The default is no limit. This directive is implemented only in version 1.37 and later.

Spool Directory = <directory> specifies the name of the directory to be used to store the spool files for this device. This directory is also used to store temporary part files when writing to a device that requires mount (USB). The default is to use the working directory.

Use LinTape = <yes|no> If yes, the archive device must use the `IBM Lintape` Kernel interface instead of the standard `ST` Linux Kernel module. The `Use MTIOCGET = No` should be used when using the Lintape module. The default is no, the use of the default `ST` Linux Kernel module is recommended.

24.4 Cloud Resource

The Cloud Storage Driver must be installed in the Bacula Storage Daemon **Plugin Directory** to be used.

Name = <Device-Name> The name of the Cloud resource. This is the logical Cloud name, and may be any string up to 127 characters in length.

Description = <Text> The description is used for display purposes as is the case with all resource.

Driver = <DriverName> This defines which driver to use. It can be S3 or Azure. There is also a File driver, which is used mostly for testing.

Host Name = <Name> This directive specifies the hostname to be used in the URL. Each Cloud service provider has a different and unique hostname. The maximum size is 255 characters and may contain a TCP port specification. This directive is not used with Azure cloud.

Bucket Name = <Name> This directive specifies the bucket name that you wish to use on the Cloud service. This name is normally a unique name that you create on the cloud service that identifies where you want to place your Cloud Volumes. The maximum bucket name size is 255 characters.

Access Key = <String> The access key is your unique user identifier given to you by your cloud service provider.

Secret Key = <String> The secret key is the security key that was given to you by your cloud service provider. It is equivalent to a password.

Protocol = <HTTP | HTTPS> The protocol defines the communications protocol to use with the cloud service provider. The two protocols currently supported are: HTTPS and HTTP. The default is HTTPS.

Uri Style = <VirtualHost | Path> This directive specifies the URI style to use to communicate with the cloud service provider. The two **Uri Styles** currently supported are: VirtualHost and Path. The default is VirtualHost.

Truncate Cache = <Truncate-kw> This directive specifies when Bacula should automatically remove (truncate) the local cache parts. Local cache parts can only be removed if they have been uploaded to the cloud. The currently implemented values are:

No Do not remove cache. With this option you must manually delete the cache parts with a `bconsole truncate cache` command, or do so with an **Admin** Job that runs an `truncate cache` command. This is the default.

AfterUpload Each part will be removed just after it is uploaded. Note, if this option[5] is specified, all restores will require a download from the Cloud.

AtEndOfJob With this option[6], at the end of the Job, every part that has been uploaded to the Cloud will be removed (truncated).

Upload = <Upload-kw> This directive specifies when local cache parts will be uploaded to the Cloud. The options are:

No Do not upload cache parts. With this option you must manually upload the cache parts with a `bconsole upload` command, or do so with an **Admin** Job that runs an `upload` command. This is the default.

EachPart With this option, each part will be uploaded when it is complete i.e. when the next part is created or at the end of the Job.

AtEndOfJob With this option[7] all parts that have not been previously uploaded will be uploaded at the end of the Job.

Maximum Concurrent Uploads = <number> The default is 3, but by using this directive, you may set it to any value you want.

Maximum Concurrent Downloads = <number> The default is 3, but by using this directive, you may set it to any value you want.

Maximum Upload Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the upload bandwidth used globally by all devices referencing this Cloud resource.

Maximum Download Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the download bandwidth used globally by all devices referencing this Cloud resource.

[5] Not yet implemented.
[6] Note yet implemented/
[7] Not implemented yet.

Region = <String> The Cloud resource can be configured to use a specific endpoint within a region. This directive is required for AWS-V4 regions. ex: **Region**="eu-central-1"

An example of a Cloud Resource might be:

```
Cloud {
  Name = S3Cloud
  Driver = "S3"
  HostName = "s3.amazonaws.com"
  BucketName = "BaculaVolumes"
  AccessKey = "BZIXAIS39DYNER5FZ"
  SecretKey = "beesheeg7iTe0Gaex7aedie4aWohfuewohGaa0"
  Protocol = HTTPS
  UriStyle = VirtualHost
  Truncate Cache = No
  Upload = EachPart
  Region = "us-east-1"
  MaximumUploadBandwidth = 5MB/s
}
```

Transfer Priority = <High | Medium | Low> When restoring directly a part from Glacier, this directive indicates the rehydration priority level. Values can be **High**, **Medium** or **Low**. Default is **High**. Those values match respectively **Expeditive**, **Standard** and **Bulk** transfers tiers within S3.

Transfer Retention = <time-period-specification> This directive indicates the number of days S3 should keep the rehydrated part online. The minimum value is 1 day. The default is 5 days.

24.5 Edit Codes for Mount and Unmount Directives

Before submitting the **Mount Command**, **Unmount Command**, **Write Part Command**, or **Free Space Command** directives to the operating system, Bacula performs character substitution of the following characters:

```
%% = %
%a = Archive device name
%e = erase (set if cannot mount and first part)
%n = part number
%m = mount point
%v = last part name (i.e. filename)
```

24.6 Devices that require a mount (USB)

All the directives in this section are implemented only in Bacula version 1.37 and later and hence are available in version 1.38.6.

As of version 1.39.5, the directives "Requires Mount", "Mount Point", "Mount Command", and "Unmount Command" apply to removable filesystems such as USB.

Requires Mount = <yes|no> You must set this directive to yes for removable devices such as USB unless they are automounted, and to no for all other devices (tapes/files). This directive indicates if the device requires to be mounted to be read, and if it must be written in a special way. If it set, **Mount Point**, **Mount Command**, **Unmount Command** directives must also be defined.

Mount Point = <directory> Directory where the device can be mounted.

Mount Command = <name-string> Command that must be executed to mount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Mount Command = "/bin/mount -t iso9660 -o ro %a %m"
```

For some media, you may need multiple commands. If so, it is recommended that you use a shell script instead of putting them all into the Mount Command. For example, instead of this:

```
Mount Command = "/usr/local/bin/mymount"
```

Where that script contains:

```
#!/bin/sh
ndasadmin enable -s 1 -o w
sleep 2
mount /dev/ndas-00323794-0p1 /backup
```

Similar consideration should be given to all other Command parameters.

Unmount Command = <name-string> Command that must be executed to unmount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount %m"
```

If you need to specify multiple commands, create a shell script.

Chapter 25

Autochanger Resource

The Autochanger resource supports single or multiple drive autochangers by grouping one or more Device resources into one unit called an autochanger in Bacula (often referred to as a "tape library" by autochanger manufacturers).

If you have an Autochanger, and you want it to function correctly, you **must** have an Autochanger resource in your Storage conf file, and your Director's Storage directives that want to use an Autochanger **must** refer to the Autochanger resource name. In previous versions of Bacula, the Director's Storage directives referred directly to Device resources that were autochangers. In version 1.38.0 and later, referring directly to Device resources will not work for Autochangers.

Name = <Autochanger-Name> Specifies the Name of the Autochanger. This name is used in the Director's Storage definition to refer to the autochanger. This directive is required.

Device = <Device-name1, device-name2, . . . > Specifies the names of the Device resource or resources that correspond to the autochanger drive. If you have a multiple drive autochanger, you must specify multiple Device names, each one referring to a separate Device resource that contains a Drive Index specification that corresponds to the drive number base zero. You may specify multiple device names on a single line separated by commas, and/or you may specify multiple Device directives. This directive is required.

Changer Device = <name-string> The specified <**name-string**> gives the system file name of the autochanger device name. If specified in this resource, the Changer Device name is not needed in the Device resource. If it is specified in the Device resource (see above), it will take precedence over one specified in the Autochanger resource.

Changer Command = <name-string> The <**name-string**> specifies an external program to be called that will automatically change volumes as required by **Bacula**. Most frequently, you will specify the Bacula supplied `mtx-changer` script as follows. If it is specified here, it need not be specified in the Device resource. If it is also specified in the Device resource, it will take precedence over the one specified in the Autochanger resource.

The following is an example of a valid Autochanger resource definition:

```
Autochanger {
  Name = "DDS-4-changer"
  Device = DDS-4-1, DDS-4-2, DDS-4-3
  Changer Device = /dev/sg0
  Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}
Device {
  Name = "DDS-4-1"
  Drive Index = 0
  Autochanger = yes
  ...
```

```
}
Device {
  Name = "DDS-4-2"
  Drive Index = 1
  Autochanger = yes
  ...
Device {
  Name = "DDS-4-3"
  Drive Index = 2
  Autochanger = yes
  Autoselect = no
  ...
}
```

Please note that it is important to include the **Autochanger = yes** directive in each Device definition that belongs to an Autochanger. A device definition should not belong to more than one Autochanger resource. Also, your Device directive in the Storage resource of the Director's conf file should have the Autochanger's resource name rather than a name of one of the Devices.

If you have a drive that physically belongs to an Autochanger but you don't want to have it automatically used when Bacula references the Autochanger for backups, for example, you want to reserve it for restores, you can add the directive:

```
Autoselect = no
```

to the Device resource for that drive. In that case, Bacula will not automatically select that drive when accessing the Autochanger. You can, still use the drive by referencing it by the Device name directly rather than the Autochanger name. An example of such a definition is shown above for the Device DDS-4-3, which will not be selected when the name DDS-4-changer is used in a Storage definition, but will be used if DDS-4-3 is used.

25.1 Capabilities

Label Media = <yes|no> If yes, permits this device to automatically label blank media without an explicit operator command. It does so by using an internal algorithm as defined on the Label Format record in each Pool resource. If this is no as by default, Bacula will label tapes only by specific operator command (`label` in the Console) or when the tape has been recycled. The automatic labeling feature is most useful when writing to disk rather than tape volumes.

Automatic mount = <yes|no> If yes (the default), permits the daemon to examine the device to determine if it contains a Bacula labeled volume. This is done initially when the daemon is started, and then at the beginning of each job. This directive is particularly important if you have set **Always Open = no** because it permits Bacula to attempt to read the device before asking the system operator to mount a tape. However, please note that the tape must be mounted before the job begins.

25.2 Messages Resource

For a description of the Messages Resource, please see the Messages Resource Chapter of this manual.

25.3 The Statistics Resource

The Statistics Resource defines the statistic collector function that can send information to a Graphite instance, to a CSV file or to bconsole with the statistics command (See **??** on page ?? for more information).

Statistics Start of the Statistics resource. Statistics directives are optional.

Name = <name> The Statistics directive **name** is used by the system administrator. This directive is required.

Description = <string> The text field contains a description of the Statistics resource that will be displayed in the graphical user interface. This directive is optional.

Interval = <time-interval> The **Intervall** directive instructs the Statistics collector thread how long it should sleep between every collection iteration. This directive is optional and the default value is 300 seconds.

Type = <CSV|Graphite> The **Type** directive specifies the Statistics backend, which may be one of the following: CSV or Graphite. This directive is required.

CSV is a simple file level backend which saves all required metrics with the following format to the file: "<time>, <metric>, <value>\n"

Where <time> is a standard Unix time (a number of seconds from 1/01/1970) with local timezone as returned by a system call time(), <metric> is a Bacula metric string and <value> is a metric value which could be in numeric format (int/float) or a string True or False for boolean variable. The CSV backend requires the **File**= parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host**= and **Port**= directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and <metricspec> is a filter which enables to use * and ? characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with ! prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this collector backend.

Example:

```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by collector to distinguish between different installations or daemons. The prefix string will be added to metric name as: "<prefix>.<metric_name>" This directive is optional.

File = <filename> The **File** is used by the CSV collector backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The collector thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If collector is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a mv shell command.

Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive Type is set to Graphite, the Host directive is required.

Host = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive Type is set to Graphite, the **Port** directive is required.

25.4 Sample Storage Daemon Configuration File

A example Storage Daemon configuration file might be the following:

```
#
# Default Bacula Storage Daemon Configuration file
#
#  For Bacula release 1.37.2 (07 July 2005) -- gentoo 1.4.16
#
# You may need to change the name of your tape drive
#   on the "Archive Device" directive in the Device
#   resource.  If you change the Name and/or the
#   "Media Type" in the Device resource, please ensure
#   that bacula-dir.conf has corresponding changes.
#
Storage {                             # definition of myself
  Name = rufus-sd
  Address = rufus
  WorkingDirectory = "$HOME/bacula/bin/working"
  Pid Directory = "$HOME/bacula/bin/working"
  Maximum Concurrent Jobs = 20
}
#
# List Directors who are permitted to contact Storage daemon
#
Director {
  Name = rufus-dir
  Password = "ZF9Ctf5PQoWCPkmR3s4atCBOusUPg+vWWyIo2VS5ti6k"
}
#
# Restricted Director, used by tray-monitor to get the
#   status of the storage daemon
#
Director {
  Name = rufus-mon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
  Monitor = yes
}
#
# Devices supported by this Storage daemon
# To connect, the Director's bacula-dir.conf must have the
#  same Name and MediaType.
#
Autochanger {
  Name = Autochanger
  Device = Drive-1
  Device = Drive-2
  Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
  Changer Device = /dev/sg0
}

Device {
  Name = Drive-1                      #
  Drive Index = 0
  Media Type = DLT-8000
  Archive Device = /dev/nst0
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
  RandomAccess = no;
  AutoChanger = yes
  Alert Command = "sh -c 'tapeinfo -f %c |grep TapeAlert|cat'"
}

Device {
  Name = Drive-2                      #
```

```
  Drive Index = 1
  Media Type = DLT-8000
  Archive Device = /dev/nst1
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
  RandomAccess = no;
  AutoChanger = yes
  Alert Command = "sh -c 'tapeinfo -f %c |grep TapeAlert|cat'"
}

Device {
  Name = "HP DLT 80"
  Media Type = DLT8000
  Archive Device = /dev/nst0
  AutomaticMount = yes;                # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
}
#Device {
#  Name = SDT-7000                     #
#  Media Type = DDS-2
#  Archive Device = /dev/nst0
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = yes;
#  RemovableMedia = yes;
#}
#Device {
#  Name = Floppy
#  Media Type = Floppy
#  Archive Device = /mnt/floppy
#  RemovableMedia = yes;
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = no;
#}
#Device {
#  Name = FileStorage
#  Media Type = File
#  Archive Device = /tmp
#  LabelMedia = yes;                   # lets Bacula label unlabeled media
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  RemovableMedia = no;
#  AlwaysOpen = no;
#}
#
# A very old Exabyte with no end of media detection
#
#Device {
#  Name = "Exabyte 8mm"
#  Media Type = "8mm"
#  Archive Device = /dev/nst0
#  Hardware end of medium = No;
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = Yes;
#  RemovableMedia = yes;
#}
#
# Send all messages to the Director,
# mount messages also are sent to the email address
#
Messages {
  Name = Standard
  director = rufus-dir = all
  operator = root = mount
}
```

Chapter 26

Messages Resource

The Messages resource defines how messages are to be handled and destinations to which they should be sent.

Even though each daemon has a full message handler, within the File daemon and the Storage daemon, you will normally choose to send all the appropriate messages back to the Director. This permits all the messages associated with a single Job to be combined in the Director and sent as a single email message to the user, or logged together in a single file.

Each message that Bacula generates (i.e. that each daemon generates) has an associated type such as INFO, WARNING, ERROR, FATAL, etc. Using the message resource, you can specify which message types you wish to see and where they should be sent. In addition, a message may be sent to multiple destinations. For example, you may want all error messages both logged as well as sent to you in an email. By defining multiple messages resources, you can have different message handling for each type of Job (e.g. Full backups versus Incremental backups).

In general, messages are attached to a Job and are included in the Job report. There are some rare cases, where this is not possible, e.g. when no job is running, or if a communications error occurs between a daemon and the director. In those cases, the message may remain in the system, and should be flushed at the end of the next Job. However, since such messages are not attached to a Job, any that are mailed will be sent to `/usr/lib/sendmail`. On some systems, such as FreeBSD, if your sendmail is in a different place, you may want to link it to the the above location.

The records contained in a Messages resource consist of a **destination** specification followed by a list of **message-types** in the format:

destination = message-type1, message-type2, message-type3, ...

or for those destinations that need and address specification (e.g. email):

destination = address = message-type1, message-type2, message-type3, ... Where **destination** is one of a predefined set of keywords that define where the message is to be sent (**stdout**, **file**, ...), **message-type** is one of a predefined set of keywords that define the type of message generated by **Bacula** (**ERROR**, **WARNING**, **FATAL**, ...), and **address** varies according to the **destination** keyword, but is typically an email address or a filename.

The following are the list of the possible record definitions that can be used in a message resource.

Messages Start of the Messages records.

Name = <name> The name of the Messages resource. The name you specify here will be used to tie this Messages resource to a Job and/or to the daemon.

MailCommand = <command> In the absence of this resource, Bacula will send all mail using the following command:

```
mail -s "Bacula Message" <recipients>
```

In many cases, depending on your machine, this command may not work. However, by using the **MailCommand**, you can specify exactly how to send the mail. During the processing of the **command** part, normally specified as a quoted string, the following substitutions will be used:

```
%% = %
%b = Job Bytes
%c = Client's name
%C = If the job is a Cloned job (Only on director side)
%d = Daemon's name (Such as host-dir or host-fd)
%D = Director's name (Also valid on file daemon)
%e = Job Exit Status
%E = Non-fatal Job Errors
%f = Job FileSet (Only on director side)
%F = Job Files
%h = Client address
%i = JobId
%I = Migration/Copy JobId (Only in Copy/Migrate Jobs)
%j = Unique Job id
%l = Job Level
%n = Job name
%o = Job Priority
%p = Pool name (Only on director side)
%P = Current PID process
%r = Recipients
%R = Read Bytes
%s = Since time
%S = Previous Job name (Only on file daemon side)
%t = Job type (Backup, ...)
%v = Volume name (Only on director side)
%w = Storage name (Only on director side)
%x = Spooling enabled? ("yes" or "no")
```

Please note: any **MailCommand** directive must be specified in the **Messages** resource **before** the desired **Mail**, **MailOnSuccess**, or **MailOnError** directive. In fact, each of those directives may be preceded by a different **MailCommand**.

The following is an example. Note, the whole command should appear on a single line in the configuration file rather than split as is done here for presentation:

mailcommand = "/opt/bacula/bin/bsmtp -h mail.example.com -f \"\(Bacula\) %r \" -s \"Bacula: %t %e of %c %l\" %r\"

The bsmtp program is provided as part of **Bacula**. For additional details, please see the **bsmtp – Customizing Your Email Messages** section (section 1.11 page 14) of the Bacula Community Edition Utility programs. Please test any **mailcommand** that you use to ensure that your bsmtp gateway accepts the addressing form that you use. Certain programs such as Exim can be very selective as to what forms are permitted particularly in the from part. Be careful, most of the samples use **%r** in the sender part of the **mailcommand**. This is a convenient way to setup the sender and the recipient at once. When you configure multiple recipients (separated by a comma) you must replace the **%r** in the sender part with only one valid email address.

OperatorCommand = <command> This resource specification is similar to the **MailCommand** except that it is used for Operator messages. The substitutions performed for the **MailCommand** are also done for this command. Normally, you will set this command to the same value as specified for the **MailCommand**. The **OperatorCommand** directive must appear in the **Messages** resource before the **Operator** directive.

<destination> = <message-type1>, <message-type2>, ... Where **destination** may be one of the following:

stdout Send the message to standard output.

stderr Send the message to standard error.

console Send the message to the console (Bacula Console). These messages are held until the console program connects to the Director.

<destination> = <address> = <message-type1>, <message-type2>, ...

Where **address** depends on the **destination**.

The **destination** may be one of the following:

director Send the message to the Director whose name is given in the **address** field. Note, in the current implementation, the Director Name is ignored, and the message is sent to the Director that started the Job.

file Send the message to the filename given in the **address** field. If the file already exists, it will be overwritten.

append Append the message to the filename given in the **address** field. If the file already exists, it will be appended to. If the file does not exist, it will be created.

syslog Send the message to the system log (syslog) using the facility specified in the **address** field. Note, for the moment, the **address** field is ignored and the message is always sent to the LOG_DAEMON facility with level LOG_ERR. See `man 3 syslog` for more details. Example:

```
syslog = all, !skipped
```

Although the **syslog** destination is not used in the default Bacula config files, in certain cases where Bacula encounters errors in trying to deliver a message, as a last resort, it will send it to the system **syslog** to prevent loss of the message, so you might occassionally check the **syslog** for Bacula output (normally `var/log/syslog`).

mail Send the message to the email addresses that are given as a comma separated list in the **address** field. Mail messages are grouped together during a job and then sent as a single email message when the job terminates. The advantage of this destination is that you are notified about every Job that runs. However, if you backup five or ten machines every night, the volume of email messages can be important. Some users use filter programs such as `procmail` to automatically file this email based on the Job termination code (see **mailcommand**).

mail on error Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates with an error condition. MailOnError messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates normally, the message is totally discarded (for this destination). If the Job terminates in error, it is emailed. By using other destinations such as **append** you can ensure that even if the Job terminates normally, the output information is saved.

mail on success Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates normally (no error condition). MailOnSuccess messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates abnormally, the message is totally discarded (for this destination). If the Job terminates normally, it is emailed.

operator Send the message to the email addresses that are specified as a comma separated list in the **address** field. This is similar to **mail** above, except that each message is sent as received. Thus there is one email per message. This is most useful for **mount** messages (see below).

console Send the message to the Bacula console.

catalog Send the message to the Catalog database. The message will be written to the table named **Log** and a timestamp field will also be added. This permits Job Reports and other messages to be recorded in the Catalog so that they can be accessed by reporting software. Bacula will prune the Log records associated with a Job when the Job records are pruned. Otherwise, Bacula never uses these records internally, so this destination is only used for special purpose programs (e.g. **Bweb**).

stdout Send the message to the standard output (normally not used).

stderr Send the message to the standard error output (normally not used).

For any destination, the **message-type** field is a comma separated list of the following types or classes of messages:

events Event messages. ex: Daemon startup/shutdown, Console login/logout, commands... This class of message is not included in the `all` message type. Specific events can be specified in the form `events.string`. Each destination directive can support up to 10 sub events.

info General information messages.

warning Warning messages. Generally this is some unusual condition but not expected to be serious.

error Non-fatal error messages. The job continues running. Any error message should be investigated as it means that something went wrong.

fatal Fatal error messages. Fatal errors cause the job to terminate.

terminate Message generated when the daemon shuts down.

notsaved Files not saved because of some error. Usually because the file cannot be accessed (i.e. it does not exist or is not mounted).

skipped Files that were skipped because of a user supplied option such as an incremental backup or a file that matches an exclusion pattern. This is not considered an error condition such as the files listed for the **notsaved** type because the configuration file explicitly requests these types of files to be skipped. For example, any unchanged file during an incremental backup, or any subdirectory if the no recursion option is specified.

mount Volume mount or intervention requests from the Storage daemon. These requests require a specific operator intervention for the job to continue.

restored The **ls** style listing generated for each file restored is sent to this message class.

saved The **ls** style listing generated for each file saved is sent to this message class.

all All message types except debug, events and saved.

security Security info/warning messages principally from unauthorized connection attempts.

alert Alert messages. These are messages generated by tape alerts.

volmgmt Volume management messages. Currently there are no volume mangement messages generated.

The following is an example of a valid Messages resource definition, where all messages except files explicitly skipped or daemon termination messages are sent by email to enforcement@sec.com. In addition all mount messages are sent to the operator (i.e. emailed to enforcement@sec.com). Finally all messages other than explicitly skipped files and files saved are sent to the console:

```
Messages {
  Name = Standard
  mail = enforcement@sec.com = all, !skipped, !terminate
  operator = enforcement@sec.com = mount
  console = all, !skipped, !saved
  catalog = all
}
```

With the exception of the email address (changed to avoid junk mail from robot's), an example Director's Messages resource is as follows. Note, the **mailcommand** and **operatorcommand** are on a single line – they had to be split for this manual:

```
Messages {
  Name = Standard
  mailcommand = "bacula/bin/bsmtp -h mail.example.com \
    -f \"\(Bacula\) %r\" -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "bacula/bin/bsmtp -h mail.example.com \
    -f \"\(Bacula\) %r\" -s \"Bacula: Intervention needed \
        for %j\" %r"
  MailOnError = security@example.com = all, !skipped, \
                !terminate
  append = "bacula/working/log" = all, !skipped, !terminate
  operator = security@example.com = mount
  console = all, !skipped, !saved
}
```

The following is an example of a valid Messages resource definition where Bacula will keep track of the user activity. It is often called "Auditing".

```
Messages {
  Name = Standard
  console = all, !skipped, !saved
  append = bacula/working/bacula.log = all

  catalog = all, events
  append = bacula/working/audit.log = events, !events.bweb
}
```

Chapter 27

Console Configuration

27.1 General

The Console configuration file is the simplest of all the configuration files, and in general, you should not need to change it except for the password. It simply contains the information necessary to contact the Director or Directors.

For a general discussion of the syntax of configuration files and their resources including the data types recognized by Bacula, please see the Configuration chapter of this manual.

The following Console Resource definition must be defined:

27.2 The Director Resource

The Director resource defines the attributes of the Director running on the network. You may have multiple Director resource specifications in a single Console configuration file. If you have more than one, you will be prompted to choose one when you start the **Console** program.

Director Start of the Director directives.

Name = <name> The director name used to select among different Directors, otherwise, this name is not used.

DIRPort = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the **--with-baseport** option of the `./configure` command. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this directive is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director.

Password = <password> Where the password is the password needed for the Director to accept the Console connection. This password must be identical to the **Password** specified in the **Director** resource of the Director's configuration file. This directive is required.

HistoryFile = <filename> Where the filename will be used to store the console command history. By default, the history file is set to `$HOME/.bconsole_history`

HistoryFileSize = <number-of-lines> Specify the history file size in lines. The default value is 100.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[1] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
  TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

[1] https://en.wikipedia.org/wiki/X.509

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

An actual example might be:

```
Director {
  Name = HeadMan
  address = rufus.cats.com
  password = xyz1erploit
}
```

27.3 The ConsoleFont Resource

The ConsoleFont resource is available only in the GNOME version of the console. It permits you to define the font that you want used to display in the main listing window.

ConsoleFont Start of the ConsoleFont directives.

Name = <name> The name of the font.

Font = <Pango Font Name> The string value given here defines the desired font. It is specified in the Pango format. For example, the default specification is:

```
Font = "LucidaTypewriter 9"
```

Thanks to Phil Stracchino for providing the code for this feature.

An different example might be:

```
ConsoleFont {
  Name = Default
  Font = "Monospace 10"
}
```

27.4 The Console Resource

There are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director resource. Typically you would use this **anonymous** console only for administrators.
- The second type of console is a "named" or "restricted" console defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.

  This second type of console begins with absolutely no privileges except those explicitly specified in the Director's Console resource. Note, the definition of what these restricted consoles can do is determined by the Director's conf file.

  Thus you may define within the Director's conf file multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges. As a default, these consoles can do absolutely nothing – no commands what so ever. You give them privileges or rather access to commands and resources by specifying access control lists in the Director's Console resource. This gives the administrator fine grained control over what particular consoles (or users) can do.
- The third type of console is similar to the above mentioned restricted console in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the **Name =** directive, is the same as a Client name, the user of that console is permitted to use the `SetIP` command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified. However, if it is specified, you can use ACLs (Access Control Lists) in the Director's configuration file to restrict the particular console (or user) to see only information pertaining to his jobs or client machine.

You may specify as many Console resources in the console's conf file. If you do so, generally the first Console resource will be used. However, if you have multiple Director resources (i.e. you want to connect to different directors), you can bind one of your Console resources to a particular Director resource, and thus when you choose a particular Director, the appropriate Console configuration resource will be used. See the "Director" directive in the Console resource described below for more information.

Note, the Console resource is optional, but can be useful for restricted consoles as noted above.

Console Start of the Console resource.

Name = <name> The Console name used to allow a restricted console to change its IP address using the SetIP command. The SetIP command must also be defined in the Director's conf CommandACL list.

Password = <password> If this password is supplied, then the password specified in the Director resource of you Console conf will be ignored. See below for more details.

Director = <director-resource-name> If this directive is specified, this Console resource will be used by bconsole when that particular director is selected when first starting bconsole. I.e. it binds a particular console resource with its name and password to a particular director.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Console to set a keepalive interval (heartbeat) in seconds on each of the sockets to communicate with the Director. It is implemented only on systems (Linux, ...) that provide the

setsockopt TCP_KEEPIDLE function. The default value is zero, which means no change is made to the socket.

CommCompression = <yes|no> If the two Bacula components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled. In the case that the compression is not effective, Bacula turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports None in the Job report.

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (bacula-fd.conf), Director resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[2] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
```

[2] https://en.wikipedia.org/wiki/X.509

```
    # the Allowed CN will be checked for this client by director
    # the client's certificate Common Name must match any of
    # the values of the Allowed CN list
    TLS Allowed CN = client1.example.com
    TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
    TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
    TLS Key = /opt/bacula/ssl/keys/director_key.pem
  }
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use `openssl`:

```
openssl dhparam -out dh4096.pem -5 4096
```

The following configuration files were supplied by Phil Stracchino. For example, if we define the following in the user's `bconsole.conf` file (or perhaps the `bwx-console.conf` file):

```
Director {
   Name = MyDirector
   DIRport = 9101
   Address = myserver
   Password = "XXXXXXXXXX"    # no, really.  this is not obfuscation.
}


Console {
   Name = restricted-user
   Password = "UntrustedUser"
}
```

Where the Password in the Director section is deliberately incorrect, and the Console resource is given a name, in this case **restricted-user**. Then in the Director's `bacula-dir.conf` file (not directly accessible by the user), we define:

```
Console {
  Name = restricted-user
  Password = "UntrustedUser"
  JobACL = "Restricted Client Save"
  ClientACL = restricted-client
```

```
  StorageACL = main-storage
  ScheduleACL = *all*
  PoolACL = *all*
  FileSetACL = "Restricted Client's FileSet"
  CatalogACL = DefaultCatalog
  CommandACL = run
}
```

the user logging into the Director from his Console will get logged in as **restricted-user**, and he will only be able to see or access a Job with the name **Restricted Client Save** a Client with the name **restricted-client**, a Storage device **main-storage**, any Schedule or Pool, a FileSet named **Restricted Client's FileSet**, a Catalog named **DefaultCatalog**, and the only command he can use in the Console is the `run` command. In other words, this user is rather limited in what he can see and do with Bacula.

The following is an example of a bconsole conf file that can access several Directors and has different Consoles depending on the director:

```
Director {
   Name = MyDirector
   DIRport = 9101
   Address = myserver
   Password = "XXXXXXXXXX"    # no, really.  this is not obfuscation.
}

Director {
   Name = SecondDirector
   DIRport = 9101
   Address = secondserver
   Password = "XXXXXXXXXX"    # no, really.  this is not obfuscation.
}

Console {
   Name = restricted-user
   Password = "UntrustedUser"
   Director = MyDirector
}

Console {
   Name = restricted-user-2
   Password = "A different UntrustedUser"
   Director = SecondDirector
}
```

The second Director referenced at "secondserver" might look like the following:

```
Console {
  Name = restricted-user-2
  Password = "A different UntrustedUser"
  JobACL = "Restricted Client Save"
  ClientACL = restricted-client
  StorageACL = second-storage
  ScheduleACL = *all*
  PoolACL = *all*
  FileSetACL = "Restricted Client's FileSet"
  CatalogACL = RestrictedCatalog
  CommandACL = run, restore
  WhereACL = "/"
}
```

To use the same Console name on both Directors, you must create two `bconsole.conf` to store the two Director/Console groups.

27.5 Console Commands

For more details on running the console and its commands, please see the **Bacula Console** chapter (chapter 1 page 1) of the Bacula Community Edition Console manual.

27.6 Sample Console Configuration File

An example Console configuration file might be the following:

```
#
# Bacula Console Configuration File
#
Director {
  Name = HeadMan
  address = "my_machine.my_domain.com"
  Password = Console_password
}
```

Chapter 28

Monitor Configuration

The Monitor configuration file is a stripped down version of the Director configuration file, mixed with a Console configuration file. It simply contains the information necessary to contact Directors, Clients, and Storage daemons you want to monitor.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual.

The following Monitor Resource definition must be defined:

- Monitor – to define the Monitor's name used to connect to all the daemons and the password used to connect to the Directors. Note, you must not define more than one Monitor resource in the Monitor configuration file.
- At least one Client, Storage or Director resource, to define the daemons to monitor.

28.1 The Monitor Resource

The Monitor resource defines the attributes of the Monitor running on the network. The parameters you define here must be configured as a Director resource in Clients and Storages configuration files, and as a Console resource in Directors configuration files.

Monitor Start of the Monitor records.

Name = <name> Specify the Director name used to connect to Client and Storage, and the Console name used to connect to Director. This record is required.

DisplayAdvancedOptions = <boolean> Display advanced options in the tray monitor (for backup and restore operations)

CommandDirectory = <directory> Directory where the tray monitor will look at a regular interval to find commands to execute.

Refresh Interval = <time> Specifies the time to wait between status requests to each daemon. It can't be set to less than 1 second, or more than 10 minutes, and the default value is 5 seconds.

28.2 The Director Resource

The Director resource defines the attributes of the Directors that are monitored by this Monitor.

As you are not permitted to define a Password in this resource, to avoid obtaining full Director privileges, you must create a Console resource in the Director's configuration file, using the Console Name and Password defined in the Monitor resource. To avoid security problems, you should configure this Console resource to allow access to no other daemons, and permit the use of only two commands: `status` and `.status` (see below for an example).

You may have multiple Director resource specifications in a single Monitor configuration file.

Director Start of the Director records.

Name = <name> The Director name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Director's configuration file. This record is required.

Port = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the **--:with-baseport** option of the `./configure` command. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this record is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director. This record is required.

28.3 The Client Resource

The Client resource defines the attributes of the Clients that are monitored by this Monitor.

You must create a Director resource in the Client's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

You may have multiple Director resource specifications in a single Monitor configuration file.

Client (or FileDaemon) Start of the Client records.

Name = <name> The Client name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Client's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula File daemon. This record is required.

Port = <port-number> Where the port is a port number at which the Bacula File daemon can be contacted. The default is 9102.

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This record is required.

28.4 The Storage Resource

The Storage resource defines the attributes of the Storages that are monitored by this Monitor.

You must create a Director resource in the Storage's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

You may have multiple Director resource specifications in a single Monitor configuration file.

Storage Start of the Storage records.

Name = <name> The Storage name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Storage's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula Storage daemon. This record is required.

Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is 9103.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This record is required.

28.5 Tray Monitor Security

There is no security problem in relaxing the permissions on `tray-monitor.conf` as long as FD, SD and DIR are configured properly, so the passwords contained in this file only gives access to the status of the daemons. It could be a security problem if you consider the status information as potentially dangerous (I don't think it is the case).

Concerning Director's configuration:
In `tray-monitor.conf`, the password in the Monitor resource must point to a restricted console in `bacula-dir.conf` (see the documentation). So, if you use this password with bconsole, you'll only have access to the status of the director (commands `status` and `.status`). It could be a security problem if there is a bug in the ACL code of the director.

Concerning File and Storage Daemons' configuration:
In `tray-monitor.conf`, the Name in the Monitor resource must point to a Director resource in `bacula-fd/sd.conf`, with the Monitor directive set to yes (once again, see the documentation). It could be a security problem if there is a bug in the code which check if a command is valid for a Monitor (this is very unlikely as the code is pretty simple).

28.6 Sample Tray Monitor configuration

An example Tray Monitor configuration file might be the following:

```
#
# Bacula Tray Monitor Configuration File
#
Monitor {
  Name = rufus-mon        # password for Directors
  RefreshInterval = 10 seconds
}

Client {
  Name = rufus-fd
  Address = rufus
  Port = 9102           # password for FileDaemon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
}
Storage {
  Name = rufus-sd
  Address = rufus
  Port = 9103           # password for StorageDaemon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
```

```
}
Director {
  Name = rufus-dir
  port = 9101
  address = rufus
}
```

28.6.1 Sample File daemon's Director record.

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#   status of the file daemon
#
Director {
  Name = rufus-mon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZ0bxT3XQxgxppTn"
  Monitor = yes
}
```

28.6.2 Sample Storage daemon's Director record.

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#   status of the storage daemon
#
Director {
  Name = rufus-mon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
  Monitor = yes
}
```

28.6.3 Sample Director's Console record.

Click here to see the full example.

```
#
# Restricted console used by tray-monitor to get the status of the director
#
Console {
  Name = Monitor
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  CommandACL = status, .status
}
```

Chapter 29

The Restore Command

29.1 General

Below, we will discuss restoring files with the Console `restore` command, which is the recommended way of doing restoring files. It is not possible to restore files by automatically starting a job as you do with Backup, Verify, ... jobs. However, in addition to the console `restore` command, there is a standalone program named `bextract`, which also permits restoring files. For more information on this program, please see the **bextract** command (command 1.6 page 5) in the Bacula Community Edition Utility programs. We don't particularly recommend the `bextract` program because it lacks many of the features of the normal Bacula restore, such as the ability to restore Win32 files to Unix systems, and the ability to restore access control lists (ACL). As a consequence, we recommend, wherever possible to use Bacula itself for restores as described below.

You may also want to look at the `bls` program in the same chapter, which allows you to list the contents of your Volumes. Finally, if you have an old Volume that is no longer in the catalog, you can restore the catalog entries using the program named `bscan`, documented in the same **bscan** command (command 1.7 page 7) in the Bacula Community Edition Utility programs.

In general, to restore a file or a set of files, you must run a `restore` job. That is a job with **Type = Restore**. As a consequence, you will need a predefined `restore` job in your `bacula-dir.conf` (Director's config) file. The exact parameters (Client, FileSet, ...) that you define are not important as you can either modify them manually before running the job or if you use the `restore` command, explained below, Bacula will automatically set them for you. In fact, you can no longer simply run a restore job. You must use the `restore` command.

Since Bacula is a network backup program, you must be aware that when you restore files, it is up to you to ensure that you or Bacula have selected the correct Client and the correct hard disk location for restoring those files. **Bacula** will quite willingly backup client A, and restore it by sending the files to a different directory on client B. Normally, you will want to avoid this, but assuming the operating systems are not too different in their file structures, this should work perfectly well, if so desired. By default, Bacula will restore data to the same Client that was backed up, and those data will be restored not to the original places but to `/tmp/bacula-restores`. You may modify any of these defaults when the `restore` command prompts you to run the job by selecting the **mod** option.

29.2 The Restore Command

Since Bacula maintains a catalog of your files and on which Volumes (disk or tape), they are stored, it can do most of the bookkeeping work, allowing you simply to specify what kind of

restore you want (current, before a particular date), and what files to restore. Bacula will then do the rest.

This is accomplished using the `restore` command in the Console. First you select the kind of restore you want, then the JobIds are selected, the File records for those Jobs are placed in an internal Bacula directory tree, and the restore enters a file selection mode that allows you to interactively walk up and down the file tree selecting individual files to be restored. This mode is somewhat similar to the standard Unix `restore` program's interactive file selection mode.

If a Job's file records have been pruned from the catalog, the `restore` command will be unable to find any files to restore. Bacula will ask if you want to restore all of them or if you want to use a regular expression to restore only a selection while reading media. See FileRegex option and below for more details on this.

Within the Console program, after entering the `restore` command, you are presented with the following selection prompt:

```
First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.
To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified Job date
    13: Select object to restore
    14: Cancel
Select item:  (1-14):
```

There are a lot of options, and as a point of reference, most people will want to slect item 5 (the most recent backup for a client). The details of the above options are:

- Item 1 will list the last 20 jobs run. If you find the Job you want, you can then select item 3 and enter its JobId(s).
- Item 2 will list all the Jobs where a specified file is saved. If you find the Job you want, you can then select item 3 and enter the JobId.
- Item 3 allows you the enter a list of comma separated JobIds whose files will be put into the directory tree. You may then select which files from those JobIds to restore. Normally, you would use this option if you have a particular version of a file that you want to restore and you know its JobId. The most common options (5 and 6) will not select a job that did not terminate normally, so if you know a file is backed up by a Job that failed (possibly because of a system crash), you can access it through this option by specifying the JobId.
- Item 4 allows you to enter any arbitrary SQL command. This is probably the most primitive way of finding the desired JobIds, but at the same time, the most flexible. Once you have found the JobId(s), you can select item 3 and enter them.
- Item 5 will automatically select the most recent Full backup and all subsequent incremental and differential backups for a specified Client. These are the Jobs and Files which, if reloaded, will restore your system to the most current saved state. It automatically enters the JobIds found into the directory tree in an optimal way such that only the most recent copy of any particular file found in the set of Jobs will be restored. This is probably the most convenient of all the above options to use if you wish to restore a selected Client to its most recent state.

There are two important things to note. First, this automatic selection will never select a job that failed (terminated with an error status). If you have such a job and want to recover one or more files from it, you will need to explicitly enter the JobId in item 3, then choose the files to restore.

If some of the Jobs that are needed to do the restore have had their File records pruned, the restore will be incomplete. Bacula currently does not correctly detect this condition. You can however, check for this by looking carefully at the list of Jobs that Bacula selects and prints. If you find Jobs with the JobFiles column set to zero, when files should have been backed up, then you should expect problems.

If all the File records have been pruned, Bacula will realize that there are no file records in any of the JobIds chosen and will inform you. It will then propose doing a full restore (non-selective) of those JobIds. This is possible because Bacula still knows where the beginning of the Job data is on the Volumes, even if it does not know where particular files are located or what their names are.

- Item 6 allows you to specify a date and time, after which Bacula will automatically select the most recent Full backup and all subsequent incremental and differential backups that started before the specified date and time.
- Item 7 allows you to specify one or more filenames (complete path required) to be restored. Each filename is entered one at a time or if you prefix a filename with the less-than symbol (<) Bacula will read that file and assume it is a list of filenames to be restored. If you prefix the filename with a question mark (?), then the filename will be interpreted as an SQL table name, and Bacula will include the rows of that table in the list to be restored. The table must contain the JobId in the first column and the FileIndex in the second column. This table feature is intended for external programs that want to build their own list of files to be restored. The filename entry mode is terminated by entering a blank line.
- Item 8 allows you to specify a date and time before entering the filenames. See Item 7 above for more details.
- Item 9 allows you find the JobIds of the most recent backup for a client. This is much like option 5 (it uses the same code), but those JobIds are retained internally as if you had entered them manually. You may then select item 11 (see below) to restore one or more directories.
- Item 10 is the same as item 9, except that it allows you to enter a before date (as with item 6). These JobIds will then be retained internally.
- Item 11 allows you to enter a list of JobIds from which you can select directories to be restored. The list of JobIds can have been previously created by using either item 9 or 10 on the menu. You may then enter a full path to a directory name or a filename preceded by a less than sign (<). The filename should contain a list of directories to be restored. All files in those directories will be restored, but if the directory contains subdirectories, nothing will be restored in the subdirectory unless you explicitly enter its name.
- Item 12 allows you to restore choosen state of the Job.
- Item 13 allows you to enter menu to select object to be restored.
- Item 14 allows you to cancel the `restore` command.

As an example, suppose that we select item 5 (restore to most recent state). If you have not specified a `client=xxx` on the command line, it it will then ask for the desired Client, which on my system, will print all the Clients found in the database as follows:

```
Defined clients:
     1: Rufus
     2: Matou
     3: Polymatou
     4: Minimatou
     5: Minou
     6: MatouVerify
     7: PmatouVerify
```

```
     8: RufusVerify
     9: Watchdog
Select Client (File daemon) resource (1-9):
```

You will probably have far fewer Clients than this example, and if you have only one Client, it will be automatically selected. In this case, I enter **Rufus** to select the Client. Then Bacula needs to know what FileSet is to be restored, so it prompts with:

```
The defined FileSet resources are:
     1: Full Set
     2: Other Files
Select FileSet resource (1-2):
```

If you have only one FileSet defined for the Client, it will be selected automatically. I choose item 1, which is my full backup. Normally, you will only have a single FileSet for each Job, and if your machines are similar (all Linux) you may only have one FileSet for all your Clients.

At this point, **Bacula** has all the information it needs to find the most recent set of backups. It will then query the database, which may take a bit of time, and it will come up with something like the following. Note, some of the columns are truncated here for presentation:

```
+-------+------+----------+-------------+-------------+------+-------+------------+
| JobId | Levl | JobFiles | StartTime   | VolumeName  | File | SesId |VolSesTime  |
+-------+------+----------+-------------+-------------+------+-------+------------+
| 1,792 | F    |  128,374 | 08-03 01:58 | DLT-19Jul02 |   67 |    18 | 1028042998 |
| 1,792 | F    |  128,374 | 08-03 01:58 | DLT-04Aug02 |    0 |    18 | 1028042998 |
| 1,797 | I    |      254 | 08-04 13:53 | DLT-04Aug02 |    5 |    23 | 1028042998 |
| 1,798 | I    |       15 | 08-05 01:05 | DLT-04Aug02 |    6 |    24 | 1028042998 |
+-------+------+----------+-------------+-------------+------+-------+------------+
You have selected the following JobId: 1792,1792,1797
Building directory tree for JobId 1792 ...
Building directory tree for JobId 1797 ...
Building directory tree for JobId 1798 ...
cwd is: /
$
```

Depending on the number of **JobFiles** for each JobId, the **Building directory tree ..."** can take a bit of time. If you notice ath all the JobFiles are zero, your Files have probably been pruned and you will not be able to select any individual files – it will be restore everything or nothing.

In our example, Bacula found four Jobs that comprise the most recent backup of the specified Client and FileSet. Two of the Jobs have the same JobId because that Job wrote on two different Volumes. The third Job was an incremental backup to the previous Full backup, and it only saved 254 Files compared to 128,374 for the Full backup. The fourth Job was also an incremental backup that saved 15 files.

Next Bacula entered those Jobs into the directory tree, with no files marked to be restored as a default, tells you how many files are in the tree, and tells you that the current working directory (`cwd`) is /. Finally, Bacula prompts with the dollar sign ($) to indicate that you may enter commands to move around the directory tree and to select files.

If you want all the files to automatically be marked when the directory tree is built, you could have entered the command `restore all`, or at the $ prompt, you can simply enter `mark *`.

Instead of choosing item 5 on the first menu (Select the most recent backup for a client), if we had chosen item 3 (Enter list of JobIds to select) and we had entered the JobIds **1792,1797,1798** we would have arrived at the same point.

One point to note, if you are manually entering JobIds, is that you must enter them in the order they were run (generally in increasing JobId order). If you enter them out of order and the same file was saved in two or more of the Jobs, you may end up with an old version of that file (i.e. not the most recent).

Directly entering the JobIds can also permit you to recover data from a Job that wrote files to tape but that terminated with an error status.

While in file selection mode, you can enter `help` or a question mark (?) to produce a summary of the available commands:

```
Command    Description
 =======    ===========
 cd         change current directory
 count      count marked files in and below the cd
 dir        long list current directory, wildcards allowed
 done       leave file selection mode
 du
 estimate   estimate restore size
 exit       same as done command
 find       find files, wildcards allowed
 help       print help
 ls         list current directory, wildcards allowed
 lsmark     list the marked files in and below the cd
 mark       mark dir/file to be restored recursively in dirs
 markdir    mark directory name to be restored (no files)
 pwd        print current working directory
 unmark     unmark dir/file to be restored recursively in dir
 unmarkdir  unmark directory name only no recursion
 quit       quit and do not do restore
 ?          print help
```

As a default no files have been selected for restore (unless you added **all** to the command line. If you want to restore everything, at this point, you should enter `mark *`, and then `done` and **Bacula** will write the bootstrap records to a file and request your approval to start a restore job.

If you do not enter the above mentioned `mark *` command, you will start with an empty slate. Now you can simply start looking at the tree and `mark` particular files or directories you want restored. It is easy to make a mistake in specifying a file to mark or unmark, and Bacula's error handling is not perfect, so please check your work by using the `ls` or `dir` commands to see what files are actually selected. Any selected file has its name preceded by an asterisk.

To check what is marked or not marked, enter the `count` command, which displays:

```
128401 total files. 128401 marked to be restored.
```

Each of the above commands will be described in more detail in the next section. We continue with the above example, having accepted to restore all files as Bacula set by default. On entering the `done` command, Bacula prints:

```
Bootstrap records written to /home/kern/bacula/working/restore.bsr
The job will require the following
   Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

   DLT-19Jul02               Tape                      DLT8000
   DLT-04Aug02               Tape                      DLT8000

128401 files selected to restore.
Run Restore job
JobName:    kernsrestore
Bootstrap:  /home/kern/bacula/working/restore.bsr
Where:      /tmp/bacula-restores
Replace:    always
FileSet:    Other Files
Client:     Rufus
Storage:    Tape
When:       2006-12-11 18:20:33
Catalog:    MyCatalog
Priority:   10
OK to run? (yes/mod/no):
```

Please examine each of the items very carefully to make sure that they are correct. In particular, look at **Where**, which tells you where in the directory structure the files will be restored, and **Client**, which tells you which client will receive the files. Note that by default the Client which will receive the files is the Client that was backed up. These items will not always be completed with the correct values depending on which of the restore options you chose. You can change any of these default items by entering **mod** and responding to the prompts.

The above assumes that you have defined a **Restore** Job resource in your Director's configuration file. Normally, you will only need one Restore Job resource definition because by its nature, restoring is a manual operation, and using the Console interface, you will be able to modify the Restore Job to do what you want.

An example Restore Job resource definition is given below.

Returning to the above example, you should verify that the Client name is correct before running the Job. However, you may want to modify some of the parameters of the restore job. For example, in addition to checking the Client it is wise to check that the Storage device chosen by Bacula is indeed correct. Although the **FileSet** is shown, it will be ignored in restore. The restore will choose the files to be restored either by reading the **Bootstrap** file, or if not specified, it will restore all files associated with the specified backup **JobId** (i.e. the JobId of the Job that originally backed up the files).

Finally before running the job, please note that the default location for restoring files is **not** their original locations, but rather the directory `/tmp/bacula-restores`. You can change this default by modifying your `bacula-dir.conf` file, or you can modify it using the **mod** option. If you want to restore the files to their original location, you must have **Where** set to nothing or to the root, i.e. `/`.

If you now enter **yes**, Bacula will run the restore Job. The Storage daemon will first request Volume **DLT-19Jul02** and after the appropriate files have been restored from that volume, it will request Volume **DLT-04Aug02**.

29.2.1 Restore a pruned job using a pattern

During a restore, if all File records are pruned from the catalog for a Job, normally Bacula can restore only all files saved. That is there is no way using the catalog to select individual files. With this new feature, Bacula will ask if you want to specify a Regex expression for extracting only a part of the full backup.

```
Building directory tree for JobId(s) 1,3 ...
There were no files inserted into the tree, so file selection
is not possible.Most likely your retention policy pruned the files

Do you want to restore all the files? (yes|no): no

Regexp matching files to restore? (empty to abort): /tmp/regress/(bin|tests)/
Bootstrap records written to /tmp/regress/working/zog4-dir.restore.1.bsr
```

See also FileRegex bsr option for more information.

29.3 Selecting Files by Filename

If you have a small number of files to restore, and you know the filenames, you can either put the list of filenames in a file to be read by Bacula, or you can enter the names one at a time. The filenames must include the full path and filename. No wild cards are used.

To enter the files, after the `restore`, you select item number 7 from the prompt list:

```
To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Cancel
Select item:  (1-12):
```

which then prompts you for the client name:

```
Defined Clients:
     1: Timmy
     2: Tibs
     3: Rufus
Select the Client (1-3): 3
```

Of course, your client list will be different, and if you have only one client, it will be automatically selected. And finally, Bacula requests you to enter a filename:

```
Enter filename:
```

At this point, you can enter the full path and filename

```
Enter filename: /home/kern/bacula/k/Makefile.in
Enter filename:
```

as you can see, it took the filename. If Bacula cannot find a copy of the file, it prints the following:

```
Enter filename: junk filename
No database record found for: junk filename
Enter filename:
```

If you want Bacula to read the filenames from a file, you simply precede the filename with a less-than symbol (<). When you have entered all the filenames, you enter a blank line, and Bacula will write the bootstrap file, tells you what tapes will be used, and proposes a Restore job to be run:

```
Enter filename:
Automatically selected Storage: DDS-4
Bootstrap records written to /home/kern/bacula/working/restore.bsr
The restore job will require the following Volumes:

   test1
1 file selected to restore.
Run Restore job
JobName:    kernsrestore
Bootstrap:  /home/kern/bacula/working/restore.bsr
Where:      /tmp/bacula-restores
Replace:    always
FileSet:    Other Files
Client:     Rufus
Storage:    DDS-4
When:       2003-09-11 10:20:53
Priority:   10
OK to run? (yes/mod/no):
```

It is possible to automate the selection by file by putting your list of files in say /tmp/file-list, then using the following command:

```
restore client=Rufus file=</tmp/file-list
```

If in modifying the parameters for the Run Restore job, you find that Bacula asks you to enter a Job number, this is because you have not yet specified either a Job number or a Bootstrap file. Simply entering zero will allow you to continue and to select another option to be modified.

29.4 Replace Options

When restoring, you have the option to specify a Replace option. This directive determines the action to be taken when restoring a file or directory that already exists. This directive can be set by selecting the **mod** option. You will be given a list of parameters to choose from. Full details on this option can be found in the Job Resource section of the Director documentation.

29.5 Command Line Arguments

If all the above sounds complicated, you will probably agree that it really isn't after trying it a few times. It is possible to do everything that was shown above, with the exception of selecting the FileSet, by using command line arguments with a single command by entering:

```
restore client=Rufus select current all done yes
```

The **client=Rufus** specification will automatically select Rufus as the client, the **current** tells Bacula that you want to restore the system to the most current state possible, the **all** selects all files, the **done** automatically finishes the file selection process, and the **yes** suppresses the final **yes/mod/no** prompt and simply runs the restore.

An example using a FileSet and a date specification:

```
restore client=Rufus fileset=Catalog before="2015-05-26 00:00:00" select all done yes
```

In this second example, the **fileset** tells Bacula which FileSet to use so you are not prompted for one, the **before** specifies a date and time to which the system should be restored. Notice that the **current** argument was removed in this example because it does not make sense when you are using the **before** argument to also specify "current"" as a date and time.

Also, when using the **before** argument, be aware that the **select** argument must come after it, otherise you will not get the results you expect.

If you wish to select specific files to restore rather than restoring the entire backup job, omit the **all**, **done**, and **yes** parameters and you will be taken to the interactive file selection prompt.

The full list of possible command line arguments is:

all – select all Files to be restored.

select – use the tree selection method.

done – do not prompt the user in tree mode.

current – automatically select the most current set of backups for the specified client.

client=xxxx – initially specifies the client from which the backup was made and the client to which the restore will be make. See also "restoreclient" keyword.

restoreclient=xxxx – if the keyword is specified, then the restore is written to that client.

jobid=nnn – specify a JobId or comma separated list of JobIds to be restored.

before="YYYY-MM-DD HH:MM:SS" – specify a date and time to which the system should be restored. Only Jobs started before the specified date/time will be selected, and as is the case for **current** Bacula will automatically find the most recent prior Full save and all Differential and Incremental saves run before the date you specify. Note, this command is not too user friendly in that you must specify the date/time exactly as shown.

file=filename – specify a filename to be restored. You must specify the full path and filename. Prefixing the entry with a less-than sign (<) will cause Bacula to assume that the filename is on your system and contains a list of files to be restored. Bacula will thus read the list from that file. Multiple file=xxx specifications may be specified on the command line.

directory=path – specify a directory to be restored. You must specify the full path. Prefixing the entry with a less-than sign (<) will cause Bacula to assume that the filename is on your system and contains a list of directories to be restored. Bacula will thus read the list from that file. Multiple directory=xxx specifications may be specified on the command line.

fileset=fileset-name – Specify a FileSet to be restored.

jobid=nnn – specify a JobId to be restored.

pool=pool-name – specify a Pool name to be used for selection of Volumes when specifying options 5 and 6 (restore current system, and restore current system before given date). This permits you to have several Pools, possibly one offsite, and to select the Pool to be used for restoring.

where=/tmp/bacula-restores – restore files in **where** directory.

yes – automatically run the restore without prompting for modifications (most useful in batch scripts).

add_prefix=/test – add a prefix to all files when restoring (like where) (can't be used with **where=**).

strip_prefix=/prod – remove a part of the filename when restoring. We suggest to not select any file outside the "strip_prefix" and explicitly "unmark" parent directories when using strip_prefix and add_prefix together to relocate files.

add_suffix=.old – add a suffix to all your files.

regexwhere=!a.pdf!a.bkp.pdf! – do complex filename manipulation like with `sed` unix command. Will overwrite other filename manipulation.

restorejob=jobname – Pre-chooses a restore job. Bacula can be configured with multiple restore jobs ("Type = Restore" in the job definition). This allows the specification of different restore properties, including a set of RunScripts. When more than one job of this type is configured, during restore, Bacula will ask for a user selection interactively, or use the given restorejob.

objectid=id – specify an Object to be restored

29.6 Using File Relocation

29.6.1 Introduction

The **where=** option is simple, but not very powerful. With file relocation, Bacula can restore a file to the same directory, but with a different name, or in an other directory without recreating the full path.

You can also do filename and path manipulations, implemented in Bacula 2.1.8 or later, such as adding a suffix to all your files, renaming files or directories, etc. Theses options will overwrite **where=** option.

For example, many users use OS snapshot features so that file `/home/eric/mbox` will be backed up from the directory `/.snap/home/eric/mbox`, which can complicate restores. If you use **where=/tmp**, the file will be restored to `/tmp/.snap/home/eric/mbox` and you will have to move the file to `/home/eric/mbox.bkp` by hand.

However, case, you could use the **strip_prefix=/.snap** and **add_suffix=.bkp** options and Bacula will restore the file to its original location – that is `/home/eric/mbox`.

To use this feature, there are command line options as described in the restore section of this manual; you can modify your restore job before running it; or you can add options to your restore job in as described in bacula-dir.conf.

```
Parameters to modify:
     1: Level
     2: Storage
    ...
    10: File Relocation
    ...
Select parameter to modify (1-12):


This will replace your current Where value
     1: Strip prefix
     2: Add prefix
     3: Add file suffix
     4: Enter a regexp
     5: Test filename manipulation
     6: Use this ?
Select parameter to modify (1-6):
```

29.6.2 RegexWhere Format

The format is very close to that used by sed or Perl (`s/replace this/by that/`) operator. A valid regexwhere expression has three fields :

- a search expression (with optionnal submatch)
- a replacement expression (with optionnal back references $1 to $9)
- a set of search options (only case-insensitive "i" at this time)

Each field is delimited by a separator specified by the user as the first character of the expression. The separator can be one of the following:

```
<separator-keyword> = / ! ; % : , ~ # = &
```

You can use several expressions separated by a commas.

Examples

Table 29.1: Regular expressions examples

Original filename	New filename	RegexWhere	Comments
c:/system.ini	c:/system.old.ini	/.ini\protect\T1\textdollar/.old.ini/	$ matches end of name
/prod/u01/pdata/	/rect/u01/rdata	/prod/rect/,/pdata/rdata/	uses two regexp
/prod/u01/pdata/	/rect/u01/rdata	!/prod/!/rect/!,/pdata/rdata/	use ! as separator
C:/WINNT	d:/WINNT	/c:/d:/i	case insensitive pattern match

29.7 Restoring Directory Attributes

Depending how you do the restore, you may or may not get the directory entries back to their original state. Here are a few of the problems you can encounter, and for same machine restores, how to avoid them.

- You backed up on one machine and are restoring to another that is either a different OS or doesn't have the same users/groups defined. Bacula does the best it can in these situations. Note, Bacula has saved the user/groups in numeric form, which means on a different machine, they may map to different user/group names.
- You are restoring into a directory that is already created and has file creation restrictions. Bacula tries to reset everything but without walking up the full chain of directories and modifying them all during the restore, which Bacula does and will not do, getting permissions back correctly in this situation depends to a large extent on your OS.
- You are doing a recursive restore of a directory tree. In this case Bacula will restore a file before restoring the file's parent directory entry. In the process of restoring the file Bacula will create the parent directory with open permissions and ownership of the file being restored. Then when Bacula tries to restore the parent directory Bacula sees that it already exists (Similar to the previous situation). If you had set the Restore job's "Replace" property to "never" then Bacula will not change the directory's permissions and ownerships to match what it backed up, you should also notice that the actual number of files restored is less then the expected number. If you had set the Restore job's "Replace" property to "always" then Bacula will change the Directory's ownership and permissions to match what it backed up, also the actual number of files restored should be equal to the expected number.
- You selected one or more files in a directory, but did not select the directory entry to be restored. In that case, if the directory is not on disk Bacula simply creates the directory with some default attributes which may not be the same as the original. If you do not select a directory and all its contents to be restored, you can still select items within the directory to be restored by individually marking those files, but in that case, you should individually use the `markdir` command to select all higher level directory entries (one at a time) to be restored if you want the directory entries properly restored.
- The `bextract` program does not restore access control lists (ACLs) to Unix machines.

29.8 Restoring on Windows

If you are restoring on WinNT/2K/XP systems, Bacula will restore the files with the original ownerships and permissions as would be expected. This is also true if you are restoring those files to an alternate directory (using the Where option in restore). However, if the alternate directory does not already exist, the Bacula File daemon (Client) will try to create it. In some cases, it may not create the directories, and if it does since the File daemon runs under the SYSTEM account, the directory will be created with SYSTEM ownership and permissions. In this case, you may have problems accessing the newly restored files.

To avoid this problem, you should create any alternate directory before doing the restore. Bacula will not change the ownership and permissions of the directory if it is already created as long as it is not one of the directories being restored (i.e. written to tape).

The default restore location is `/tmp/bacula-restores/` and if you are restoring from drive `E:`, the default will be `/tmp/bacula-restores/e/`, so you should ensure that this directory exists before doing the restore, or use the **mod** option to select a different **where** directory that does exist.

Some users have experienced problems restoring files that participate in the Active Directory. They also report that changing the userid under which Bacula (bacula-fd.exe) runs, from SYSTEM to a Domain Admin userid, resolves the problem.

29.9 Restoring Files Can Be Slow

Restoring files is generally **much** slower than backing them up for several reasons. The first is that during a backup the tape is normally already positioned and Bacula only needs to write. On the other hand, because restoring files is done so rarely, Bacula keeps only the start file and block on the tape for the whole job rather than on a file by file basis which would use quite a lot of space in the catalog.

Bacula will forward space to the correct file mark on the tape for the Job, then forward space to the correct block, and finally sequentially read each record until it gets to the correct one(s) for the file or files you want to restore. Once the desired files are restored, Bacula will stop reading the tape.

Finally, instead of just reading a file for backup, during the restore, Bacula must create the file, and the operating system must allocate disk space for the file as Bacula is restoring it.

For all the above reasons the restore process is generally much slower than backing up (sometimes it takes three times as long).

29.10 Problems Restoring Files

The most frequent problems users have restoring files are error messages such as:

```
04-Jan 00:33 z217-sd: RestoreFiles.2005-01-04_00.31.04 Error:
block.c:868 Volume data error at 20:0! Short block of 512 bytes on
device /dev/tape discarded.
```

or

```
04-Jan 00:33 z217-sd: RestoreFiles.2005-01-04_00.31.04 Error:
block.c:264 Volume data error at 20:0! Wanted ID: "BB02", got ".".
Buffer discarded.
```

Both these kinds of messages indicate that you were probably running your tape drive in fixed block mode rather than variable block mode. Fixed block mode will work with any program that reads tapes sequentially such as tar, but Bacula repositions the tape on a block basis when restoring files because this will speed up the restore by orders of magnitude when only a few files are being restored. There are several ways that you can attempt to recover from this unfortunate situation.

Try the following things, each separately, and reset your Device resource to what it is now after each individual test:

1. Set "Block Positioning = no" in your Device resource and try the restore. This is a new directive and untested.
2. Set "Minimum Block Size = 512" and "Maximum Block Size = 512" and try the restore. If you are able to determine the block size your drive was previously using, you should try that size if 512 does not work. This is a really horrible solution, and it is not at all recommended to continue backing up your data without correcting this condition. Please see the Tape Testing chapter for more on this.
3. Try editing the restore.bsr file at the Run xxx yes/mod/no prompt before starting the restore job and remove all the VolBlock statements. These are what causes Bacula to reposition the tape, and where problems occur if you have a fixed block size set for your drive. The VolFile commands also cause repositioning, but this will work regardless of the block size.
4. Use bextract to extract the files you want – it reads the Volume sequentially if you use the include list feature, or if you use a .bsr file, but remove all the VolBlock statements after the .bsr file is created (at the Run yes/mod/no) prompt but before you start the restore.

29.11 Restore Errors

There are a number of reasons why there may be restore errors or warning messages. Some of the more common ones are:

file count mismatch This can occur for the following reasons:

- You requested Bacula not to overwrite existing or newer files.
- A Bacula miscount of files/directories. This is an on-going problem due to the complications of directories, soft/hard link, and such. Simply check that all the files you wanted were actually restored.

file size error When Bacula restores files, it checks that the size of the restored file is the same as the file status data it saved when starting the backup of the file. If the sizes do not agree, Bacula will print an error message. This size mismatch most often occurs because the file was being written as Bacula backed up the file. In this case, the size that Bacula restored will be greater than the status size. This often happens with log files.

If the restored size is smaller, then you should be concerned about a possible tape error and check the Bacula output as well as your system logs.

29.12 Example Restore Job Resource

```
Job {
  Name = "RestoreFiles"
  Type = Restore
  Client = Any-client
  FileSet = "Any-FileSet"
```

```
   Storage = Any-storage
   Where = /tmp/bacula-restores
   Messages = Standard
   Pool = Default
}
```

If **Where** is not specified, the default location for restoring files will be their original locations.

29.13 File Selection Commands

After you have selected the Jobs to be restored and Bacula has created the in-memory directory tree, you will enter file selection mode as indicated by the dollar sign (**$**) prompt. While in this mode, you may use the commands listed above. The basic idea is to move up and down the in memory directory structure with the `cd` command much as you normally do on the system. Once you are in a directory, you may select the files that you want restored. As a default no files are marked to be restored. If you wish to start with all files, simply enter: `cd /` and `mark *`. Otherwise proceed to select the files you wish to restore by marking them with the `mark` command. The available commands are:

cd The `cd` command changes the current directory to the argument specified. It operates much like the Unix `cd` command. Wildcard specifications are not permitted.

Note, on Windows systems, the various drives (c:, d:, ...) are treated like a directory within the file tree while in the file selection mode. As a consequence, you must do a `cd c:` or possibly in some cases a `cd C:` (note upper case) to get down to the first directory.

dir The `dir` command is similar to the `ls` command, except that it prints it in long format (all details). This command can be a bit slower than the `ls` command because it must access the catalog database for the detailed information for each file.

estimate The `estimate` command prints a summary of the total files in the tree, how many are marked to be restored, and an estimate of the number of bytes to be restored. This can be useful if you are short on disk space on the machine where the files will be restored.

find The `find` command accepts one or more arguments and displays all files in the tree that match that argument. The argument may have wildcards. It is somewhat similar to the Unix command `find / -name arg`.

ls The `ls` command produces a listing of all the files contained in the current directory much like the Unix `ls` command. You may specify an argument containing wildcards, in which case only those files will be listed.

Any file that is marked to be restored will have its name preceded by an asterisk (*****). Directory names will be terminated with a forward slash (**/**) to distinguish them from filenames.

du The `du` command is similar to the `ls` command but it also prints size of each file directory listed. For directory it is just a sum of all files sizes in all directories below.

lsmark The `lsmark` command is the same as the `ls` except that it will print only those files marked for extraction. The other distinction is that it will recursively descend into any directory selected.

mark The `mark` command allows you to mark files to be restored. It takes a single argument which is the filename or directory name in the current directory to be marked for extraction. The argument may be a wildcard specification, in which case all files that match in the current directory are marked to be restored. If the argument matches a directory rather than a file, then the directory and all the files contained in that directory (recursively) are marked to be restored. Any marked file will have its name preceded with an asterisk (*****) in the output produced by the `ls` or `dir` commands. Note, supplying a full path on the mark command does not work as expected to select a file or directory in the current directory.

Also, the `mark` command works on the current and lower directories but does not touch higher level directories.

After executing the `mark` command, it will print a brief summary:

```
No files marked.
```

If no files were marked, or:

```
nn files marked.
```

if some files are marked.

unmark The `unmark` is identical to the `mark` command, except that it unmarks the specified file or files so that they will not be restored. Note: the `unmark` command works from the current directory, so it does not unmark any files at a higher level. First do a `cd /` before the `unmark *` command if you want to unmark everything.

pwd The `pwd` command prints the current working directory. It accepts no arguments.

count The `count` command prints the total files in the directory tree and the number of files marked to be restored.

done This command terminates file selection mode.

exit This command terminates file selection mode (the same as done).

quit This command terminates the file selection and does not run the restore job.

help This command prints a summary of the commands available.

? This command is the same as the `help` command.

If your filename contains some weird caracters, you can use ?, * or \\. For example, if your filename contains a \, you can use \\\\.

```
* mark weird_file\\\\with-backslash
```

29.14 Restoring When Things Go Wrong

This and the following sections will try to present a few of the kinds of problems that can come up making restoring more difficult. We will try to provide a few ideas how to get out of these problem situations. In addition to what is presented here, there is more specific information on restoring a Client and your Server in the Disaster Recovery Using Bacula chapter of this manual.

Problem My database is broken.

Solution For either `MySQL` or `PostgreSQL`, see the vendor's documentation. They have specific tools that check and repair databases, see the database repair sections of this manual for links to vendor information.

Assuming the above does not resolve the problem, you will need to restore or rebuild your catalog. Note, if it is a matter of some inconsistencies in the Bacula tables rather than a broken database, then running the **dbcheck** command (command 1.12 page 15)[1] might help, but you will need to ensure that your database indexes are properly setup. Please see the Database Performance Issues sections of this manual for more details.

Problem How do I restore my catalog?

[1] Bacula Community Edition Utility programs

Solution with a Catalog backup If you have backed up your database nightly (as you should) and you have made a bootstrap file, you can immediately load back your database (or the ASCII SQL output). Make a copy of your current database, then re-initialize it, by running the following scripts:

```
./drop_bacula_tables
./make_bacula_tables
```

After re-initializing the database, you should be able to run Bacula. If you now try to use the `restore` command, it will not work because the database will be empty. However, you can manually run a restore job and specify your bootstrap file. You do so by entering the `run` command in the console and selecting the restore job. If you are using the default `bacula-dir.conf`, this Job will be named **RestoreFiles**. Most likely it will prompt you with something such as:

```
Run Restore job
JobName:    RestoreFiles
Bootstrap:  /home/kern/bacula/working/restore.bsr
Where:      /tmp/bacula-restores
Replace:    always
FileSet:    Full Set
Client:     rufus-fd
Storage:    File
When:       2005-07-10 17:33:40
Catalog:    MyCatalog
Priority:   10
OK to run? (yes/mod/no):
```

A number of the items will be different in your case. What you want to do is: to use the mod option to change the Bootstrap to point to your saved bootstrap file; and to make sure all the other items such as Client, Storage, Catalog, and Where are correct. The FileSet is not used when you specify a bootstrap file. Once you have set all the correct values, run the Job and it will restore the backup of your database, which is most likely an ASCII dump.

You will then need to follow the instructions for your database type to recreate the database from the ASCII backup file. See the Catalog Maintenance chapter of this manual for examples of the command needed to restore a database from an ASCII dump (they are shown in the Compacting Your XXX Database sections).

Also, please note that after you restore your database from an ASCII backup, you do NOT want to do a `make_bacula_tables` command, or you will probably erase your newly restored database tables.

Solution with a Job listing If you did save your database but did not make a bootstrap file, then recovering the database is more difficult. You will probably need to use bextract to extract the backup copy. First you should locate the listing of the job report from the last catalog backup. It has important information that will allow you to quickly find your database file. For example, in the job report for the CatalogBackup shown below, the critical items are the Volume name(s), the Volume Session Id and the Volume Session Time. If you know those, you can easily restore your Catalog.

```
22-Apr 10:22 HeadMan: Start Backup JobId 7510,
Job=CatalogBackup.2005-04-22_01.10.0
22-Apr 10:23 HeadMan: \bacula{} 1.37.14 (21Apr05): 22-Apr-2005 10:23:06
  JobId:                  7510
  Job:                    CatalogBackup.2005-04-22_01.10.00
  Backup Level:           Full
  Client:                 Polymatou
  FileSet:                "CatalogFile" 2003-04-10 01:24:01
  Pool:                   "Default"
  Storage:                "DLTDrive"
  Start time:             22-Apr-2005 10:21:00
  End time:               22-Apr-2005 10:23:06
  FD Files Written:       1
  SD Files Written:       1
  FD Bytes Written:       210,739,395
  SD Bytes Written:       210,739,521
  Rate:                   1672.5 KB/s
```

```
Software Compression:   None
Volume name(s):         DLT-22Apr05
Volume Session Id:      11
Volume Session Time:    1114075126
Last Volume Bytes:      1,428,240,465
Non-fatal FD errors:    0
SD Errors:              0
FD termination status:  OK
SD termination status:  OK
Termination:            Backup OK
```

From the above information, you can manually create a bootstrap file, and then follow the instructions given above for restoring your database. A reconstructed bootstrap file for the above backup Job would look like the following:

```
Volume="DLT-22Apr05"
VolSessionId=11
VolSessionTime=1114075126
FileIndex=1-1
```

Where we have inserted the Volume name, Volume Session Id, and Volume Session Time that correspond to the values in the job report. We've also used a FileIndex of one, which will always be the case providing that there was only one file backed up in the job.

The disadvantage of this bootstrap file compared to what is created when you ask for one to be written, is that there is no File and Block specified, so the restore code must search all data in the Volume to find the requested file. A fully specified bootstrap file would have the File and Blocks specified as follows:

```
Volume="DLT-22Apr05"
VolSessionId=11
VolSessionTime=1114075126
VolFile=118-118
VolBlock=0-4053
FileIndex=1-1
```

Once you have restored the ASCII dump of the database, you will then to follow the instructions for your database type to recreate the database from the ASCII backup file. See the Catalog Maintenance chapter of this manual for examples of the command needed to restore a database from an ASCII dump (they are shown in the Compacting Your XXX Database sections).

Also, please note that after you restore your database from an ASCII backup, you do NOT want to do a `make_bacula_tables` command, or you will probably erase your newly restored database tables.

Solution without a Job Listing If you do not have a job listing, then it is a bit more difficult. Either you use the **bscan** program (program 1.7 page 7) to scan the contents of your tape into a database, which can be very time consuming depending on the size of the tape, or you can use the **bls** program (program 1.5 page 2) to list everything on the tape, and reconstruct a bootstrap file from the bls listing for the file or files you want following the instructions given above.

There is a specific example of how to use `bls` below.

Problem I try to restore the last known good full backup by specifying item 3 on the restore menu then the JobId to restore. Bacula then reports:

```
1 Job 0 Files
```

and restores nothing.

Solution Most likely the File records were pruned from the database either due to the File Retention period expiring or by explicitly purging the Job. By using the `llist jobid=nn` command, you can obtain all the important information about the job:

```
llist jobid=120
           JobId: 120
             Job: save.2005-12-05_18.27.33
         Job.Name: save
      PurgedFiles: 0
             Type: B
            Level: F
     Job.ClientId: 1
      Client.Name: Rufus
        JobStatus: T
        SchedTime: 2005-12-05 18:27:32
        StartTime: 2005-12-05 18:27:35
          EndTime: 2005-12-05 18:27:37
         JobTDate: 1133803657
     VolSessionId: 1
   VolSessionTime: 1133803624
         JobFiles: 236
        JobErrors: 0
  JobMissingFiles: 0
       Job.PoolId: 4
        Pool.Name: Full
    Job.FileSetId: 1
  FileSet.FileSet: BackupSet
```

Then you can find the Volume(s) used by doing:

```
sql
select VolumeName from JobMedia,Media where JobId=1 and JobMedia.MediaId=Media.MediaId;
```

Finally, you can create a bootstrap file as described in the previous problem above using this information.

If you are using Bacula version 1.38.0 or greater, when you select item 3 from the menu and enter the JobId, it will ask you if you would like to restore all the files in the job, and it will collect the above information and write the bootstrap file for you.

Problem You don't have a bootstrap file, and you don't have the Job report for the backup of your database, but you did backup the database, and you know the Volume to which it was backed up.

Solution Either bscan the tape (see below for bscanning), or better use `bls` to find where it is on the tape, then use `bextract` to restore the database. For example,

```
./bls -j -V DLT-22Apr05 /dev/nst0
```

Might produce the following output:

```
bls: butil.c:258 Using device: "/dev/nst0" for reading.
21-Jul 18:34 bls: Ready to read from volume "DLT-22Apr05" on device "DLTDrive"
(/dev/nst0).
Volume Record: File:blk=0:0 SessId=11 SessTime=1114075126 JobId=0 DataLen=164
...
Begin Job Session Record: File:blk=118:0 SessId=11 SessTime=1114075126
JobId=7510
   Job=CatalogBackup.2005-04-22_01.10.0 Date=22-Apr-2005 10:21:00 Level=F Type=B
End Job Session Record: File:blk=118:4053 SessId=11 SessTime=1114075126
JobId=7510
   Date=22-Apr-2005 10:23:06 Level=F Type=B Files=1 Bytes=210,739,395 Errors=0
Status=T
...
21-Jul 18:34 bls: End of Volume at file 201 on device "DLTDrive" (/dev/nst0),
Volume "DLT-22Apr05"
21-Jul 18:34 bls: End of all volumes.
```

Of course, there will be many more records printed, but we have indicated the essential lines of output. From the information on the Begin Job and End Job Session Records, you can reconstruct a bootstrap file such as the one shown above.

Problem How can I find where a file is stored.

Solution Normally, it is not necessary, you just use the `restore` command to restore the most recently saved version (menu option 5), or a version saved before a given date (menu option 8). If you know the JobId of the job in which it was saved, you can use menu option 3 to enter that JobId.

If you would like to know the JobId where a file was saved, select restore menu option 2.

You can also use the `query` command to find information such as:

```
*query
Available queries:
     1: List up to 20 places where a File is saved regardless of the
directory
     2: List where the most recent copies of a file are saved
     3: List last 20 Full Backups for a Client
     4: List all backups for a Client after a specified time
     5: List all backups for a Client
     6: List Volume Attributes for a selected Volume
     7: List Volumes used by selected JobId
     8: List Volumes to Restore All Files
     9: List Pool Attributes for a selected Pool
    10: List total files/bytes by Job
    11: List total files/bytes by Volume
    12: List Files for a selected JobId
    13: List Jobs stored on a selected MediaId
    14: List Jobs stored for a given Volume name
    15: List Volumes \bacula{} thinks are in changer
    16: List Volumes likely to need replacement from age or errors
Choose a query (1-16):
```

Problem I didn't backup my database. What do I do now?

Solution This is probably the worst of all cases, and you will probably have to re-create your database from scratch and then bscan in all your Volumes, which is a very long, painful, and inexact process.

There are basically three steps to take:

1. Ensure that your SQL server is running (`MySQL` or `PostgreSQL`) and that the Bacula database (normally bacula) exists. See the Installation chapter of the manual.
2. Ensure that the Bacula databases are created. This is also described at the above link.
3. Start and stop the Bacula Director using the propriate `bacula-dir.conf` file so that it can create the Client and Storage records which are not stored on the Volumes. Without these records, scanning is unable to connect the Job records to the proper client.

When the above is complete, you can begin bscanning your Volumes. Please see the **bscan** section (section 1.7 page 7) of the Bacula Community Edition Utility programs.

s

Chapter 30

Automatic Volume Recycling

By default, once Bacula starts writing a Volume, it may append to the volume, but it will not overwrite the existing data thus destroying it. However when Bacula **recycles** a Volume, the Volume becomes available for being reused, and Bacula can at some later time overwrite the previous contents of that Volume. At that point all previous data on that Volume will be lost. If the Volume is a tape, the tape will be rewritten from the beginning. If the Volume is a disk file, the file will be truncated before being rewritten.

You may not want Bacula to automatically recycle (reuse) Volumes. Doing so may require a large number of Volumes though. However, it is also possible to manually recycle Volumes so that they may be reused. For more on manual recycling, see the section entitled Manually Recycling Volumes below in this chapter.

Most people prefer to have a Pool of Volumes that are used for daily backups and recycled once a week, another Pool of Volumes that are used for Full backups once a week and recycled monthly, and finally a Pool of Volumes that are used once a month and recycled after a year or two. With a scheme like this, the number of Volumes in your pool or pools remains constant.

By properly defining your Volume Pools with appropriate Retention periods, Bacula can manage the recycling (such as defined above) automatically.

Automatic recycling of Volumes is controlled by four records in the `Pool` resource definition in the Director's configuration file. These four records are:

- AutoPrune = yes
- VolumeRetention = <time>
- Recycle = yes
- RecyclePool = <APool>
- ScratchPool = <APool>

The above first three directives are all you need assuming that you fill each of your Volumes then wait the Volume Retention period before reusing them, providing there is some non-pruned Jobs or Files on the Volume. **Recycle Pool** and **Scratch Pool** directives are not required. If not defined, Bacula will recycle Volumes and keep them in the current Pool.

If you want Bacula to stop using a Volume and recycle it before it is full, you will need to use one or more additional directives such as:

- **Use Volume Once**=yes
- **Volume Use Duration**=ttt
- **Maximum Volume Jobs**=nnn

- **Maximum Volume Bytes**=mmm

Please see below and the Basic Volume Management chapter of this manual for more complete examples.

Automatic recycling of Volumes is performed by Bacula only when it wants a new Volume and no appendable Volumes are available in the Pool. It will then search the Pool for any Volumes with the *Recycle* flag set and the Volume Status is *Purged*. At that point, it will choose the oldest purged volume and recycle it.

If there are no volumes with status *Purged*, then the recycling occurs in two steps:

The first is that the Catalog for a Volume must be pruned of all Jobs (i.e. Purged) and Files contained on that Volume.

The second step is the actual recycling of the Volume. Only Volumes marked *Full* or *Used* will be considered for pruning. The Volume will be purged, all Jobs and Files associated to this Volume are pruned from Catalog, if the **VolumeRetention** period has expired. When a Volume is marked as *Purged*, it means that no Catalog records for Jobs and Files reference that Volume, and the Volume can be recycled and reused. Please note a Volume can be reused even though the **Volume Retention** period has not expired if the Jobs and Files associated to the Volume had been already pruned. Until recycling actually occurs, the Volume data remains intact. If no Volumes can be found for recycling for any of the reasons stated above, Bacula will request operator intervention (i.e. it will ask you to label a new volume).

A key point mentioned above, that can be a source of frustration, is that Bacula will only recycle purged Volumes if there is no other appendable Volume available, otherwise, it will always write to an appendable Volume before recycling even if there are Volumes marked as *Purged*. This preserves your data as long as possible. So, if you wish to "force" Bacula to use a purged Volume, you must first ensure that no other Volume in the Pool is marked *Append*. If necessary, you can manually set a volume to *Full*. The reason for this is that Bacula wants to preserve the data on your old Volumes (even though purged from the catalog) as long as absolutely possible before overwriting it. There are also a number of directives such as **Volume Use Duration** that will automatically mark a volume as *Used* and thus no longer appendable.

30.1 Automatic Pruning

As Bacula writes files to a Volume, it keeps a list of files, jobs, and volumes in a database called the catalog. Among other things, the database helps Bacula to decide which files to back up in an incremental or differential backup, and helps you locate files on past backups when you want to restore something. However, the catalog will grow larger and larger as time goes on, and eventually it can become unacceptably large.

Bacula's process for removing entries from the catalog is called Pruning. The default is Automatic Pruning, which means that once a Job record reaches a certain age (e.g. 30 days old) and a pruning occurs, it will be removed from the catalog. Note that Job records that are required for current restore won't be removed automatically, and File records are needed for VirtualFull and Accurate backups. Once a job has been pruned, you can still restore it from the backup Volume, provided that the Volume has not been recycled, but one additional step is required: scanning the volume with `bscan`. The alternative to Automatic Pruning is Manual Pruning, in which you explicitly tell Bacula to erase the catalog entries for a volume. You'd usually do this when you want to reuse a Bacula volume, because there's no point in keeping a list of files that USED TO BE on a volume. Or, if the catalog is starting to get too big, you could prune the oldest jobs to save space. Manual pruning is done with the **prune** command (command 1.5 page 11) in the Bacula Community Edition Console manual (thanks to Bryce Denney for the above explanation).

30.2 Pruning Directives

There are three pruning durations. All apply to catalog database Jobs and Files records and not to the actual data in a Volume. The pruning (or retention) durations are for: Volumes (Jobs and Files records in the Volume), Jobs (Job records), and Files (File records). The durations inter-depend a bit because if Bacula prunes a Volume, it automatically removes all the Job records, and, consequently, all the File records. Also when a Job record is pruned, all the File records for that Job are also pruned (deleted) from the catalog.

Having the File records in the database means that you can examine all the files backed up for a particular Job. They take the most space in the catalog (probably 90-95% of the total). When the File records are pruned, the Job records can remain, and you can still examine what Jobs ran, but not the details of the Files backed up. In addition, without the File records, you cannot use the Console `restore` command to restore specific files.

When a Job record is pruned, the Volume (Media record) for that Job can still remain in the database, and if you do a `list volumes`, you will see the volume information, but the Job records (and its File records) will no longer be available.

In each case, pruning removes information about where older files are, but it also prevents the catalog from growing to be too large. You choose the retention periods in function of how many files you are backing up and the time periods you want to keep those records online, and the size of the database. You can always re-insert the records (with 98% of the original data) by using `bscan` to scan in a whole Volume or any part of the volume that you want.

By setting **AutoPrune** to yes you will permit Bacula to automatically prune Volumes in the Pool when a Job needs a Volume. Volume pruning means removing Jobs and Files records from the catalog. It does not shrink the size of the Volume or affect the Volume data until the Volume gets overwritten. When a Job requests a volume and there are no Volumes with Volume Status *Append*, *Recycle* or *Purged* available, Bacula will begin volume pruning. This means that all Jobs that are older than the **VolumeRetention** period will be pruned from every Volume that has Volume Status *Full* or *Used* and has Recycle set to yes. Pruning consists of deleting the corresponding Job, File, and JobMedia records from the catalog database. No change to the physical data on the Volume occurs during the pruning process. When all files are pruned from a Volume (i.e. no records in the catalog), the Volume will be marked as *Purged* implying that no Jobs remain on the volume. The Pool records that control the pruning are described below.

AutoPrune = <yes|no> If AutoPrune is set to yes (default), Bacula will automatically apply the Volume retention period when running a Job and it needs a new Volume but no appendable volumes are available. At that point, Bacula will prune all Volumes that can be pruned (i.e. AutoPrune set) in an attempt to find a usable volume. If during the autoprune, all files are pruned from the Volume, it will be marked with VolStatus *Purged*. The default is yes. Note, that although the File and Job records may be pruned from the catalog, a Volume will be marked Purged (and hence ready for recycling) if the Volume status is Append, Full, Used, or Error. If the Volume has another status, such as Archive, Read-Only, Disabled, Busy, or Cleaning, the Volume status will not be changed to Purged.

Volume Retention = <time-period-specification> The Volume Retention record defines the length of time that Bacula will guarantee that the Volume is not reused (i.e. recycled). As long as a Volume has the status **Append**, it will not be recycled even though the retention period may have expired, until the status is changed to some other status such as Full, Used, Purged, ...

The retention period when applied starts at the time of the last write to the Volume. For a Volume to recycled, the volume must not have a status of **Append** and the retention period must have expired. Even in that case, the Volume will not be recycled if any other appendable Volume is available for the Job to complete. As noted above, the Volume status will be marked **Purged** by Bacula prior to it being recycled.

Note, when all the Job records that are on the Volume have been removed, the Volume will be marked *Purged* (i.e. it has no more valid Jobs stored on it), and the Volume may

be recycled even if the **Volume Retention** period has not expired.

When this time period expires, and if **AutoPrune** is set to yes, and a new Volume is needed, but no appendable Volume is available, Bacula will prune (remove) Job records that are older than the specified Volume Retention period even if the Job or File retention has not been reached. Normally this should not happen since the Volume Retention period should always be set greater than the Job Retention period, which should be greater than the File Retention period.

The Volume Retention period takes precedence over any Job Retention period you have specified in the Client resource. It should also be noted, that the Volume Retention period is obtained by reading the Catalog Database Media record rather than the Pool resource record. This means that if you change the VolumeRetention in the Pool resource record (in bacula-dir.conf), you must ensure that the corresponding change is made in the catalog by using the `update pool` command. Doing so will insure that any new Volumes will be created with the changed Volume Retention period. Any existing Volumes will have their own copy of the Volume Retention period that can only be changed on a Volume by Volume basis using the `update volume` command.

When all Job catalog entries are removed from the volume, its VolStatus is set to *Purged*. The files remain physically on the Volume until the volume is overwritten.

Retention periods are specified in seconds, minutes, hours, days, weeks, months, quarters, or years on the record. See the Configuration chapter of this manual for additional details of time specification.

The default Volume Retention period is 1 year.

Recycle = <yes|no> This statement tells Bacula whether or not the particular Volume can be recycled (i.e. rewritten). If Recycle is set to no (the default), then even if Bacula prunes all the Jobs on the volume and it is marked *Purged*, it will not consider the volume for recycling. If Recycle is set to yes and all Jobs have been pruned, the volume status will be set to *Purged* and the volume may then be reused when another volume is needed. If the volume is reused, it is relabeled with the same Volume Name, however all previous data will be lost.

It is also possible to "force" pruning of all Volumes in the Pool associated with a Job by adding **Prune Files**=yes to the Job resource.

30.3 Recycling Algorithm

When a Job needs a Volume and no appendable, recycled or purged is available, Bacula starts the recycling algorithm pruning Jobs and Files for the oldest Volume in the pool used by the job. After the Volume is pruned, and if the **Recycle** flag is on (**Recycle**=yes) for that Volume, Bacula will relabel it and write new data on it.

As mentioned above, there are two key points for getting a Volume to be recycled. First, the Volume must no longer be marked Append (there are a number of directives to automatically make this change), and second since the last write on the Volume, one or more of the Retention periods must have expired so that there are no more catalog backup job records that reference that Volume. Once both those conditions are satisfied, the volume can be marked Purged and hence recycled.

The full algorithm that Bacula uses when it needs a new Volume is:

The algorithm described below assumes that AutoPrune is enabled, that Recycling is turned on, and that you have defined appropriate Retention periods, or used the defaults for all these items.

- If the request is for an Autochanger device, look only for Volumes in the Autochanger (i.e. with InChanger set and that have the correct Storage device).

- Search the Pool for a Volume with VolStatus=Append (if there is more than one, the Volume with the oldest date last written is chosen. If two have the same date then the one with the lowest MediaId is chosen).
- Search the Pool for a Volume with VolStatus=Recycle and the InChanger flag is set true (if there is more than one, the Volume with the oldest date last written is chosen. If two have the same date then the one with the lowest MediaId is chosen).
- Search the Pool for a Volume with VolStatus=Purged and try recycling any purged Volumes.
- Prune volumes applying Volume retention period (Volumes with VolStatus *Full* or *Used* are pruned). Note, when all the File and Job records are pruned from a Volume, the Volume will be marked *Purged* and this may be prior to the expiration of the Volume retention period.
- Prune the oldest Volume if **RecycleOldestVolume**=yes (the Volume with the oldest LastWritten date and VolStatus equal to *Full* or *Used* is chosen). This record ensures that all retention periods are properly respected.
- Purge the oldest Volume if PurgeOldestVolume=yes (the Volume with the oldest LastWritten date and VolStatus equal to *Full* or *Used* is chosen). We strongly recommend against the use of **PurgeOldestVolume** as it can quite easily lead to loss of current backup data.
- Search for a Volume in the Scratch Pool (the default Scratch pool or another Scratch Pool defined for the pool) and if found move it to the current Pool for the Job and use it. Note, when the Scratch Volume is moved into the current Pool, the basic Pool defaults are applied as if it is a newly labeled Volume (equivalent to an `update volume from pool` command).
- If we were looking for Volumes in the Autochanger, go back to step 2 above, but this time, look for any Volume whether or not it is in the Autochanger.
- Attempt to create a new Volume if automatic labeling is enabled and the maximum number of Volumes is not reached.
- Give up and ask operator.

The above occurs when Bacula has finished writing a Volume or when no Volume is present in the drive.

On the other hand, if you have inserted a different Volume after the last job, and Bacula recognizes the Volume as valid, the Storage daemon will request authorization from the Director to use this Volume. In this case, if you have set **Recycle Current Volume = yes**=and the Volume is marked as Used or Full, Bacula will prune the volume and if all jobs were removed during the pruning (respecting the retention periods), the Volume will be recycled and re-used.

If you want to strictly observe the **Volume Retention Period** under all circumstances – i.e. you have a Volume with critical data, you must set **Recycle**=no. However, in doing so, you must manually do the recycling of the Volume for it to be used again.

The recycling algorithm in this case is:

- If the VolStatus is *Append* or *Recycle* is set, the volume will be used.
- If **Recycle Current Volume** is set and the volume is marked *Full* or *Used*, Bacula will prune the volume (applying the retention period). If all Jobs are pruned from the volume, it will be recycled.

This permits users to manually change the Volume every day and load volumes in an order different from what is in the catalog, and if the volume does not contain a current copy of your backup data, it will be used.

A few points to keep in mind:

1. If a pool doesn't have maximum volumes defined then Bacula will prefer to demand new volumes over forcibly purging older volumes.
2. If volumes become free through pruning, then they get marked as *Purged* and are immediately available for recycling – these will be used in preference to creating new volumes.
3. If the Job, File, and Volume retention periods are different, then it's common to see a volume with no files or jobs listed in the database, but which is still not marked as *Purged*, if **Autoprune** is set to no in the pool resource.

30.4 Recycle Status

Each Volume inherits the Recycle status (yes or no) from the Pool resource record when the Media record is created (normally when the Volume is labeled). This Recycle status is stored in the Media record of the Catalog. Using the Console program, you may subsequently change the Recycle status for each Volume. For example in the following output from `list volumes`:

```
+----------+-------+--------+---------+------------+--------+-----+
| VolumeNa | Media | VolSta | VolByte | LastWritte | VolRet | Rec |
+----------+-------+--------+---------+------------+--------+-----+
| File0001 | File  | Full   | 4190055 | 2002-05-25 | 14400  | 1   |
| File0002 | File  | Full   | 1896460 | 2002-05-26 | 14400  | 1   |
| File0003 | File  | Full   | 1896460 | 2002-05-26 | 14400  | 1   |
| File0004 | File  | Full   | 1896460 | 2002-05-26 | 14400  | 1   |
| File0005 | File  | Full   | 1896460 | 2002-05-26 | 14400  | 1   |
| File0006 | File  | Full   | 1896460 | 2002-05-26 | 14400  | 1   |
| File0007 | File  | Purged | 1896466 | 2002-05-26 | 14400  | 1   |
+----------+-------+--------+---------+------------+--------+-----+
```

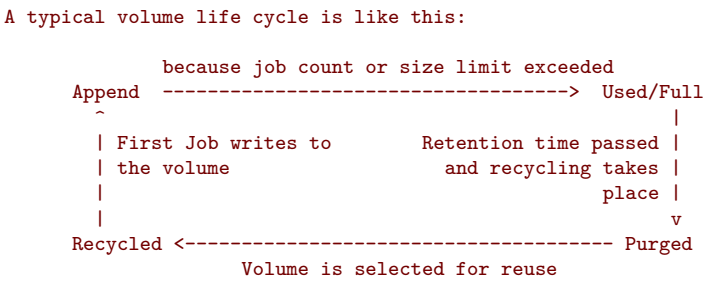
all the volumes are marked as recyclable, and the last Volume, File0007 has been purged, so it may be immediately recycled. The other volumes are all marked recyclable and when their Volume Retention period (14400 seconds or four hours) expires, they will be eligible for pruning, and possibly recycling. Even though Volume File0007 has been purged, all the data on the Volume is still recoverable. A purged Volume simply means that there are no entries in the Catalog. Even if the Volume Status is changed to *Recycle*, the data on the Volume will be recoverable. The data is lost only when the Volume is re-labeled and re-written.

To modify Volume File0001 so that it cannot be recycled, you use the `update volume pool=File` command in the console program, or simply `update` and Bacula will prompt you for the information.

```
+----------+------+-------+---------+-------------+-------+-----+
| VolumeNa | Media| VolSta| VolByte | LastWritten | VolRet| Rec |
+----------+------+-------+---------+-------------+-------+-----+
| File0001 | File | Full  | 4190055 | 2002-05-25  | 14400 | 0   |
| File0002 | File | Full  | 1897236 | 2002-05-26  | 14400 | 1   |
| File0003 | File | Full  | 1896460 | 2002-05-26  | 14400 | 1   |
| File0004 | File | Full  | 1896460 | 2002-05-26  | 14400 | 1   |
| File0005 | File | Full  | 1896460 | 2002-05-26  | 14400 | 1   |
| File0006 | File | Full  | 1896460 | 2002-05-26  | 14400 | 1   |
| File0007 | File | Purged| 1896466 | 2002-05-26  | 14400 | 1   |
+----------+------+-------+---------+-------------+-------+-----+
```

In this case, **File0001** will never be automatically recycled. The same effect can be achieved by setting the Volume Status to Read-Only.

As you have noted, the Volume Status (VolStatus) column in the catalog database contains the current status of the Volume, which is normally maintained automatically by Bacula. To give you an idea of some of the values it can take during the life cycle of a Volume, here is a picture of the process.



30.5 Making Bacula Use a Single Tape

Most people will want Bacula to fill a tape and when it is full, a new tape will be mounted, and so on. However, as an extreme example, it is possible for Bacula to write on a single tape, and every night to rewrite it. To get this to work, you must do two things: first, set the **VolumeRetention** to less than your save period (one day), and the second item is to make Bacula mark the tape as full after using it once. This is done using **UseVolumeOnce**=yes. If this latter record is not used and the tape is not full after the first time it is written, Bacula will simply append to the tape and eventually request another volume. Using the tape only once, forces the tape to be marked *Full* after each use, and the next time Bacula runs, it will recycle the tape.

An example Pool resource that does this is:

```
Pool {
  Name = DDS-4
  Use Volume Once = yes
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 12h # expire after 12 hours
  Recycle = yes
}
```

30.6 Daily, Weekly, Monthly Tape Usage Example

This example is meant to show you how one could define a fixed set of volumes that Bacula will rotate through on a regular schedule. There are an infinite number of such schemes, all of which have various advantages and disadvantages. Note: these volumes may either be tape volumes or disk volumes.

We start with the following assumptions:

- A single tape has more than enough capacity to do a full save.
- There are ten tapes that are used on a daily basis for incremental backups. They are prelabeled Daily1 ... Daily10.
- There are four tapes that are used on a weekly basis for full backups. They are labeled Week1 ... Week4.
- There are 12 tapes that are used on a monthly basis for full backups. They are numbered Month1 ... Month12
- A full backup is done every Saturday evening (tape inserted Friday evening before leaving work).
- No backups are done over the weekend (this is easy to change).
- The first Friday of each month, a Monthly tape is used for the Full backup.

- Incremental backups are done Monday - Friday (actually Tue-Fri mornings).

We start the system by doing a Full save to one of the weekly volumes or one of the monthly volumes. The next morning, we remove the tape and insert a Daily tape. Friday evening, we remove the Daily tape and insert the next tape in the Weekly series. Monday, we remove the Weekly tape and re-insert the Daily tape. On the first Friday of the next month, we insert the next Monthly tape in the series rather than a Weekly tape, then continue. When a Daily tape finally fills up, Bacula will request the next one in the series, and the next day when you notice the email message, you will mount it and Bacula will finish the unfinished incremental backup.

What does this give? Well, at any point, you will have the last complete Full save plus several Incremental saves. For any given file you want to recover (or your whole system), you will have a copy of that file every day for at least the last 14 days. For older versions, you will have at least three and probably four Friday full saves of that file, and going back further, you will have a copy of that file made on the beginning of the month for at least a year.

So you have copies of any file (or your whole system) for at least a year, but as you go back in time, the time between copies increases from daily to weekly to monthly.

What would the Bacula configuration look like to implement such a scheme?

```
Schedule {
  Name = "NightlySave"
  Run = Level=Full Pool=Monthly 1st sat at 03:05
  Run = Level=Full Pool=Weekly 2nd-5th sat at 03:05
  Run = Level=Incremental Pool=Daily tue-fri at 03:05
}
Job {
  Name = "NightlySave"
  Type = Backup
  Level = Full
  Client = LocalMachine
  FileSet = "File Set"
  Messages = Standard
  Storage = DDS-4
  Pool = Daily
  Schedule = "NightlySave"
}
# Definition of file storage device
Storage {
  Name = DDS-4
  Address = localhost
  SDPort = 9103
  Password = XXXXXXXXXXXX
  Device = FileStorage
  Media Type = 8mm
}
FileSet {
  Name = "File Set"
  Include {
    Options { signature=MD5 }
    File = ffffffffffffffff
  }
  Exclude  { File=*.o }
}
Pool {
  Name = Daily
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 10d   # recycle in 10 days
  Maximum Volumes = 10
  Recycle = yes
}
Pool {
  Name = Weekly
  Use Volume Once = yes
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 30d  # recycle in 30 days (default)
```

```
  Recycle = yes
}
Pool {
  Name = Monthly
  Use Volume Once = yes
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 365d  # recycle in 1 year
  Recycle = yes
}
```

30.7 Automatic Pruning and Recycling Example

Perhaps the best way to understand the various resource records that come into play during automatic pruning and recycling is to run a Job that goes through the whole cycle. If you add the following resources to your Director's configuration file:

```
Schedule {
  Name = "30 minute cycle"
  Run = Level=Full Pool=File Messages=Standard Storage=File
        hourly at 0:05
  Run = Level=Full Pool=File Messages=Standard Storage=File
        hourly at 0:35
}
Job {
  Name = "Filetest"
  Type = Backup
  Level = Full
  Client=XXXXXXXXXX
  FileSet="Test Files"
  Messages = Standard
  Storage = File
  Pool = File
  Schedule = "30 minute cycle"
}
# Definition of file storage device
Storage {
  Name = File
  Address = XXXXXXXXXX
  SDPort = 9103
  Password = XXXXXXXXXXXX
  Device = FileStorage
  Media Type = File
}
FileSet {
  Name = "File Set"
  Include {
    Options { signature=MD5 }
    File = fffffffffffffff
  }
  Exclude  { File=*.o }
}
Pool {
  Name = File
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = "File"
  AutoPrune = yes
  VolumeRetention = 4h
  Maximum Volumes = 12
  Recycle = yes
}
```

Where you will need to replace the **fffffffff**'s by the appropriate files to be saved for your configuration. For the FileSet Include, choose a directory that has one or two megabytes maximum since there will probably be approximately eight copies of the directory that Bacula will cycle through.

In addition, you will need to add the following to your Storage daemon's configuration file:

```
Device {
  Name = FileStorage
  Media Type = File
  Archive Device = /tmp
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
```

With the above resources, Bacula will start a Job every half hour that saves a copy of the directory you chose to /tmp/File0001 ... /tmp/File0012. After 4 hours, Bacula will start recycling the backup Volumes (/tmp/File0001 ...). You should see this happening in the output produced. Bacula will automatically create the Volumes (Files) the first time it uses them.

To turn it off, either delete all the resources you've added, or simply comment out the `Schedule` record in the `Job` resource.

30.8 Manually Recycling Volumes

Although automatic recycling of Volumes is implemented in version 1.20 and later (see the Automatic Recycling of Volumes chapter of this manual), you may want to manually force reuse (recycling) of a Volume.

Assuming that you want to keep the Volume name, but you simply want to write new data on the tape, the steps to take are:

- Use the `update volume` command in the Console to ensure that the *Recycle* field is set to 1
- Use the `purge jobs volume` command in the Console to mark the Volume as *Purged*. Check by using `list volumes`.

Once the Volume is marked Purged, it will be recycled the next time a Volume is needed.

If you wish to reuse the volume by giving it a new name, follow the following steps:

- Use the `purge jobs volume` command in the Console to mark the Volume as *Purged*. Check by using `list volumes`.
- Use the Console `relabel` command to relabel the Volume.

Please note that the `relabel` command applies only to tape Volumes.

For Bacula versions prior to 1.30 or to manually relabel the Volume, use the instructions below:

- Use the `delete volume` command in the Console to delete the Volume from the Catalog.
- If a different tape is mounted, use the `unmount` command, remove the tape, and insert the tape to be renamed.
- Write an EOF mark in the tape using the following commands:

  ```
  mt -f /dev/nst0 rewind
  mt -f /dev/nst0 weof
  ```

  where you replace `/dev/nst0` with the appropriate device name on your system.
- Use the `label` command to write a new label to the tape and to enter it in the catalog.

Please be aware that the `delete` command can be dangerous. Once it is done, to recover the File records, you must either restore your database as it was before the `delete` command, or use the `bscan` utility program to scan the tape and recreate the database entries.

Chapter 31

Basic Volume Management

This chapter presents most all the features needed to do Volume management. Most of the concepts apply equally well to both tape and disk Volumes. However, the chapter was originally written to explain backing up to disk, so you will see it is slanted in that direction, but all the directives presented here apply equally well whether your volume is disk or tape.

If you have a lot of hard disk storage or you absolutely must have your backups run within a small time window, you may want to direct Bacula to backup to disk Volumes rather than tape Volumes. This chapter is intended to give you some of the options that are available to you so that you can manage either disk or tape volumes.

31.1 Key Concepts and Resource Records

Getting Bacula to write to disk rather than tape in the simplest case is rather easy. In the Storage daemon's configuration file, you simply define an **Archive Device** to be a directory. For example, if you want your disk backups to go into the directory `/home/bacula/backups`, you could use the following:

```
Device {
  Name = FileBackup
  Media Type = File
  Archive Device = /home/bacula/backups
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
```

Assuming you have the appropriate **Storage** resource in your Director's configuration file that references the above Device resource,

```
Storage {
  Name = FileStorage
  Address = ...
  Password = ...
  Device = FileBackup
  Media Type = File
}
```

Bacula will then write the archive to the file **/home/bacula/backups/**<**volume-name**> where <volume-name> is the volume name of a Volume defined in the Pool. For example, if you have labeled a Volume named **Vol001**, Bacula will write to the file `/home/bacula/backups/Vol001`.

Although you can later move the archive file to another directory, you should not rename it or it will become unreadable by Bacula. This is because each archive has the filename as part of the internal label, and the internal label must agree with the system filename before Bacula will use it.

Although this is quite simple, there are a number of problems. The first is that unless you specify otherwise, Bacula will always write to the same volume until you run out of disk space. This problem is addressed below.

In addition, if you want to use concurrent jobs that write to several different volumes at the same time, you will need to understand a number of other details. An example of such a configuration is given at the end of this chapter under Concurrent Disk Jobs.

31.1.1 Pool Options to Limit the Volume Usage

Some of the options you have, all of which are specified in the Pool record, are:

- To write each Volume only once (i.e. one Job per Volume or file in this case), use:
  UseVolumeOnce = yes.
- To write nnn Jobs to each Volume, use:
  Maximum Volume Jobs = nnn.
- To limit the maximum size of each Volume, use:
  Maximum Volume Bytes = mmmm.
  Note, if you use disk volumes and do not specify a Maximum Volume Bytes, Bacula will write to your first Volume until your whole disk fills. This is probably not what you want. We recommend keeping your total number of Volumes created in the Catalog down to less than 5,000 for performance reasons when pruning. For a small site (10-20 clients), you might choose a Maximum Volume Bytes of 5G, for a medium site (20-100 clients) you might choose as size of 50G, and for large sites (greater than 100 clients) 200G.
  The advantage of bigger Volumes is that it improves performance during pruning (smaller number of Volumes to prune). The advantage of smaller Volumes is that you waste less disk space by being able to recycle them more often than big Volumes.
  As of Bacula version 2.0 and greater large Volumes present little performance penality during restores, because Bacula is able to seek to the start of each Job (and often within a Job).
- To limit the use time (i.e. write the Volume for a maximum of five days), use:
  Volume Use Duration = ttt.

Note that although you probably would not want to limit the number of bytes on a tape as you would on a disk Volume, the other options can be very useful in limiting the time Bacula will use a particular Volume (be it tape or disk). For example, the above directives can allow you to ensure that you rotate through a set of daily Volumes if you wish.

As mentioned above, each of those directives is specified in the Pool or Pools that you use for your Volumes. In the case of **Maximum Volume Job**, **Maximum Volume Bytes**, and **Volume Use Duration**, you can actually specify the desired value on a Volume by Volume basis. The value specified in the Pool record becomes the default when labeling new Volumes. Once a Volume has been created, it gets its own copy of the Pool defaults, and subsequently changing the Pool will have no effect on existing Volumes. You can either manually change the Volume values, or refresh them from the Pool defaults using the `update volume` command in the Console. As an example of the use of one of the above, suppose your Pool resource contains:

```
Pool {
  Name = File
  Pool Type = Backup
```

```
  Volume Use Duration = 23h
}
```

then if you run a backup once a day (every 24 hours), Bacula will use a new Volume for each backup, because each Volume it writes can only be used for 23 hours after the first write. Note, setting the use duration to 23 hours is not a very good solution for tapes unless you have someone on-site during the weekends, because Bacula will want a new Volume and no one will be present to mount it, so no weekend backups will be done until Monday morning.

31.1.2 Automatic Volume Labeling

Use of the above records brings up another problem – that of labeling your Volumes. For automated disk backup, you can either manually label each of your Volumes, or you can have Bacula automatically label new Volumes when they are needed. While, the automatic Volume labeling in version 1.30 and prior is a bit simplistic, but it does allow for automation, the features added in version 1.31 permit automatic creation of a wide variety of labels including information from environment variables and special Bacula Counter variables.

Please note that automatic Volume labeling can also be used with tapes, but it is not nearly so practical since the tapes must be pre-mounted. This requires some user interaction. Automatic labeling from templates does NOT work with autochangers since Bacula will not access unknown slots. There are several methods of labeling all volumes in an autochanger magazine. For more information on this, please see the Autochanger chapter of this manual.

Automatic Volume labeling is enabled by making a change to both the Pool resource (Director) and to the Device resource (Storage daemon) shown above. In the case of the Pool resource, you must provide Bacula with a label format that it will use to create new names. In the simplest form, the label format is simply the Volume name, to which Bacula will append a four digit number. This number starts at 0001 and is incremented for each Volume the catalog contains. Thus if you modify your Pool resource to be:

```
Pool {
  Name = File
  Pool Type = Backup
  Volume Use Duration = 23h
  LabelFormat = "Vol"
}
```

Bacula will create Volume names Vol0001, Vol0002, and so on when new Volumes are needed. Much more complex and elaborate labels can be created using variable expansion defined in the **Variable Expansion** chapter (chapter 2 page 17) of the Bacula Community Edition Miscellaneous Guide.

The second change that is necessary to make automatic labeling work is to give the Storage daemon permission to automatically label Volumes. Do so by adding **LabelMedia = yes** to the Device resource as follows:

```
Device {
  Name = File
  Media Type = File
  Archive Device = /home/bacula/backups
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
  LabelMedia = yes
}
```

You can find more details of the **Label Format** Pool record in Label Format description of the Pool resource records.

31.1.3 Restricting the Number of Volumes and Recycling

Automatic labeling discussed above brings up the problem of Volume management. With the above scheme, a new Volume will be created every day. If you have not specified Retention periods, your Catalog will continue to fill keeping track of all the files Bacula has backed up, and this procedure will create one new archive file (Volume) every day.

The tools Bacula gives you to help automatically manage these problems are the following:

1. Catalog file record retention periods, the File Retention = ttt record in the Client resource.
2. Catalog job record retention periods, the Job Retention = ttt record in the Client resource.
3. The AutoPrune = yes record in the Client resource to permit application of the above two retention periods.
4. The Volume Retention = ttt record in the Pool resource.
5. The AutoPrune = yes record in the Pool resource to permit application of the Volume retention period.
6. The Recycle = yes record in the Pool resource to permit automatic recycling of Volumes whose Volume retention period has expired.
7. The Recycle Oldest Volume = yes record in the Pool resource tells Bacula to Prune the oldest volume in the Pool, and if all files were pruned to recycle this volume and use it.
8. The Recycle Current Volume = yes record in the Pool resource tells Bacula to Prune the currently mounted volume in the Pool, and if all files were pruned to recycle this volume and use it.
9. The Purge Oldest Volume = yes record in the Pool resource permits a forced recycling of the oldest Volume when a new one is needed. **N.B. This record ignores retention periods! We highly recommend not to use this record, but instead use Recycle Oldest Volume**
10. The Maximum Volumes = nnn record in the Pool resource to limit the number of Volumes that can be created.

The first three directives (**File Retention**, **Job Retention**, and **AutoPrune**) determine the amount of time that Job and File records will remain in your Catalog, and they are discussed in detail in the Automatic Volume Recycling chapter of this manual.

Volume Retention, **AutoPrune**, and **Recycle** determine how long Bacula will keep Jobs and Files associated to your Volumes before reusing them, and they are also discussed in detail in the Automatic Volume Recycling chapter of this manual.

The Maximum Volumes record can also be used in conjunction with the Volume Retention period to limit the total number of archive Volumes (files) that Bacula will create. By setting an appropriate Volume Retention period, a Volume will be purged just before it is needed and thus Bacula can cycle through a fixed set of Volumes. Cycling through a fixed set of Volumes can also be done by setting **Recycle Oldest Volume = yes** or **Recycle Current Volume = yes**. In this case, when Bacula needs a new Volume, it will prune the specified volume.

31.2 Concurrent Disk Jobs

Above, we discussed how you could have a single device named **FileBackup** that writes to volumes in `/home/bacula/backups`. You can, in fact, run multiple concurrent jobs using the

Storage definition given with this example, and all the jobs will simultaneously write into the Volume that is being written.

Now suppose you want to use multiple Pools, which means multiple Volumes, or suppose you want each client to have its own Volume and perhaps its own directory such as /home/bacula/client1 and /home/bacula/client2 ... With the single Storage and Device definition above, neither of these two is possible. Why? Because Bacula disk storage follows the same rules as tape devices. Only one Volume can be mounted on any Device at any time. If you want to simultaneously write multiple Volumes, you will need multiple Device resources in your bacula-sd.conf file, and thus multiple Storage resources in your bacula-dir.conf.

OK, so now you should understand that you need multiple Device definitions in the case of different directories or different Pools, but you also need to know that the catalog data that Bacula keeps contains only the Media Type and not the specific storage device. This permits a tape for example to be re-read on any compatible tape drive. The compatibility being determined by the Media Type. The same applies to disk storage. Since a volume that is written by a Device in say directory /home/bacula/backups cannot be read by a Device with an Archive Device definition of /home/bacula/client1, you will not be able to restore all your files if you give both those devices **Media Type = File**. During the restore, Bacula will simply choose the first available device, which may not be the correct one. If this is confusing, just remember that the Directory has only the Media Type and the Volume name. It does not know the **Archive Device** (or the full path) that is specified in the Storage daemon. Thus you must explicitly tie your Volumes to the correct Device by using the Media Type.

The example shown below shows a case where there are two clients, each using its own Pool and storing their Volumes in different directories.

31.3 An Example

The following example is not very practical, but can be used to demonstrate the proof of concept in a relatively short period of time. The example consists of a two clients that are backed up to a set of 12 archive files (Volumes) for each client into different directories on the Storage machine. Each Volume is used (written) only once, and there are four Full saves done every hour (so the whole thing cycles around after three hours).

What is key here is that each physical device on the Storage daemon has a different Media Type. This allows the Director to choose the correct device for restores ...

The Director's configuration file is as follows:

```
Director {
  Name = my-dir
  QueryFile = "~/bacula/bin/query.sql"
  PidDirectory = "~/bacula/working"
  WorkingDirectory = "~/bacula/working"
  Password = dir_password
}
Schedule {
  Name = "FourPerHour"
  Run = Level=Full hourly at 0:05
  Run = Level=Full hourly at 0:20
  Run = Level=Full hourly at 0:35
  Run = Level=Full hourly at 0:50
}
Job {
  Name = "RecycleExample"
  Type = Backup
  Level = Full
  Client = Rufus
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = FileStorage
```

```
  Pool = Recycle
  Schedule = FourPerHour
}

Job {
  Name = "RecycleExample2"
  Type = Backup
  Level = Full
  Client = Roxie
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = FileStorage1
  Pool = Recycle1
  Schedule = FourPerHour
}

FileSet {
  Name = "Example FileSet"
  Include {
        Options {
        compression=GZIP
        signature=SHA1
                }
    File = /home/kern/bacula/bin
         }
}

Client {
  Name = Rufus
  Address = rufus
  Catalog = BackupDB
  Password = client_password
}

Client {
  Name = Roxie
  Address = roxie
  Catalog = BackupDB
  Password = client1_password
}

Storage {
  Name = FileStorage
  Address = rufus
  Password = local_storage_password
  Device = RecycleDir
  Media Type = File
}

Storage {
  Name = FileStorage1
  Address = rufus
  Password = local_storage_password
  Device = RecycleDir1
  Media Type = File1
}

Catalog {
  Name = BackupDB
  dbname = bacula; user = bacula; password = ""
}
Messages {
  Name = Standard
  ...
}
Pool {
  Name = Recycle
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = "Recycle-"
  AutoPrune = yes
  VolumeRetention = 2h
  Maximum Volumes = 12
  Recycle = yes
}
```

```
Pool {
  Name = Recycle1
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = "Recycle1-"
  AutoPrune = yes
  VolumeRetention = 2h
  Maximum Volumes = 12
  Recycle = yes
}
```

and the Storage daemon's configuration file is:

```
Storage {
  Name = my-sd
  WorkingDirectory = "~/bacula/working"
  Pid Directory = "~/bacula/working"
  MaximumConcurrentJobs = 10
}
Director {
  Name = my-dir
  Password = local_storage_password
}
Device {
  Name = RecycleDir
  Media Type = File
  Archive Device = /home/bacula/backups
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}

Device {
  Name = RecycleDir1
  Media Type = File1
  Archive Device = /home/bacula/backups1
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}

Messages {
  Name = Standard
  director = my-dir = all
}
```

With a little bit of work, you can change the above example into a weekly or monthly cycle (take care about the amount of archive disk space used).

31.4 Backing up to Multiple Disks

Bacula can, of course, use multiple disks, but in general, each disk must be a separate Device specification in the Storage daemon's conf file, and you must then select what clients to backup to each disk. You will also want to give each Device specification a different Media Type so that during a restore, Bacula will be able to find the appropriate drive.

The situation is a bit more complicated if you want to treat two different physical disk drives (or partitions) logically as a single drive, which Bacula does not directly support. However, it is possible to back up your data to multiple disks as if they were a single drive by linking the Volumes from the first disk to the second disk.

For example, assume that you have two disks named /disk1 and /disk2. If you then create a standard Storage daemon Device resource for backing up to the first disk, it will look like the following:

```
Device {
  Name = client1
  Media Type = File
  Archive Device = /disk1
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
```

Since there is no way to get the above Device resource to reference both /disk1 and /disk2 we do it by pre-creating Volumes on /disk2 with the following:

```
ln -s /disk2/Disk2-vol001 /disk1/Disk2-vol001
ln -s /disk2/Disk2-vol002 /disk1/Disk2-vol002
ln -s /disk2/Disk2-vol003 /disk1/Disk2-vol003
...
```

At this point, you can label the Volumes as Volume **Disk2-vol001**, **Disk2-vol002**, ... and Bacula will use them as if they were on /disk1 but actually write the data to /disk2. The only minor inconvenience with this method is that you must explicitly name the disks and cannot use automatic labeling unless you arrange to have the labels exactly match the links you have created.

An important thing to know is that Bacula treats disks like tape drives as much as it can. This means that you can only have a single Volume mounted at one time on a disk as defined in your Device resource in the Storage daemon's conf file. You can have multiple concurrent jobs running that all write to the one Volume that is being used, but if you want to have multiple concurrent jobs that are writing to separate disks drives (or partitions), you will need to define separate Device resources for each one, exactly as you would do for two different tape drives. There is one fundamental difference, however. The Volumes that you create on the two drives cannot be easily exchanged as they can for a tape drive, because they are physically resident (already mounted in a sense) on the particular drive. As a consequence, you will probably want to give them different Media Types so that Bacula can distinguish what Device resource to use during a restore. An example would be the following:

```
Device {
  Name = Disk1
  Media Type = File1
  Archive Device = /disk1
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}

Device {
  Name = Disk2
  Media Type = File2
  Archive Device = /disk2
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
```

With the above device definitions, you can run two concurrent jobs each writing at the same time, one to /disk1 and the other to /disk2. The fact that you have given them different

Media Types will allow Bacula to quickly choose the correct Storage resource in the Director when doing a restore.

31.5 Considerations for Multiple Clients

If we take the above example and add a second Client, here are a few considerations:

- Although the second client can write to the same set of Volumes, you will probably want to write to a different set.
- You can write to a different set of Volumes by defining a second Pool, which has a different name and a different **LabelFormat**.
- If you wish the Volumes for the second client to go into a different directory (perhaps even on a different filesystem to spread the load), you would do so by defining a second Device resource in the Storage daemon. The **Name** must be different, and the **Archive Device** could be different. To ensure that Volumes are never mixed from one pool to another, you might also define a different MediaType (e.g. **File1**).

In this example, we have two clients, each with a different Pool and a different number of archive files retained. They also write to different directories with different Volume labeling.

The Director's configuration file is as follows:

```
Director {
  Name = my-dir
  QueryFile = "~/bacula/bin/query.sql"
  PidDirectory = "~/bacula/working"
  WorkingDirectory = "~/bacula/working"
  Password = dir_password
}
# Basic weekly schedule
Schedule {
  Name = "WeeklySchedule"
  Run = Level=Full fri at 1:30
  Run = Level=Incremental sat-thu at 1:30
}
FileSet {
  Name = "Example FileSet"
  Include {
        Options {
        compression=GZIP
        signature=SHA1
                }
    File = /home/kern/bacula/bin
         }
}
Job {
  Name = "Backup-client1"
  Type = Backup
  Level = Full
  Client = client1
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = File1
  Pool = client1
  Schedule = "WeeklySchedule"
}
Job {
  Name = "Backup-client2"
  Type = Backup
  Level = Full
  Client = client2
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = File2
```

```
  Pool = client2
  Schedule = "WeeklySchedule"
}
Client {
  Name = client1
  Address = client1
  Catalog = BackupDB
  Password = client1_password
  File Retention = 7d
}
Client {
  Name = client2
  Address = client2
  Catalog = BackupDB
  Password = client2_password
}
# Two Storage definitions with different Media Types
#  permits different directories
Storage {
  Name = File1
  Address = rufus
  Password = local_storage_password
  Device = client1
  Media Type = File1
}
Storage {
  Name = File2
  Address = rufus
  Password = local_storage_password
  Device = client2
  Media Type = File2
}
Catalog {
  Name = BackupDB
  dbname = bacula; user = bacula; password = ""
}
Messages {
  Name = Standard
  ...
}
# Two pools permits different cycling periods and Volume names
# Cycle through 15 Volumes (two weeks)
Pool {
  Name = client1
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = "Client1-"
  AutoPrune = yes
  VolumeRetention = 13d
  Maximum Volumes = 15
  Recycle = yes
}
# Cycle through 8 Volumes (1 week)
Pool {
  Name = client2
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = "Client2-"
  AutoPrune = yes
  VolumeRetention = 6d
  Maximum Volumes = 8
  Recycle = yes
}
```

and the Storage daemon's configuration file is:

```
Storage {
  Name = my-sd
  WorkingDirectory = "~/bacula/working"
  Pid Directory = "~/bacula/working"
  MaximumConcurrentJobs = 10
}
Director {
```

```
  Name = my-dir
  Password = local_storage_password
}
# Archive directory for Client1
Device {
  Name = client1
  Media Type = File1
  Archive Device = /home/bacula/client1
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
# Archive directory for Client2
Device {
  Name = client2
  Media Type = File2
  Archive Device = /home/bacula/client2
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
Messages {
  Name = Standard
  director = my-dir = all
}
```

Chapter 32

Automated Disk Backup

If you manage five or ten machines and have a nice tape backup, you don't need Pools, and you may wonder what they are good for. In this chapter, you will see that Pools can help you optimize disk storage space. The same techniques can be applied to a shop that has multiple tape drives, or that wants to mount various different Volumes to meet their needs.

The rest of this chapter will give an example involving backup to disk Volumes, but most of the information applies equally well to tape Volumes.

32.1 The Problem

A site that I administer (a charitable organization) had a tape DDS-3 tape drive that was failing. The exact reason for the failure is still unknown. Worse yet, their full backup size is about 15GB whereas the capacity of their broken DDS-3 was at best 8GB (rated 6/12). A new DDS-4 tape drive and the necessary cassettes was more expensive than their budget could handle.

32.2 The Solution

They want to maintain six months of backup data, and be able to access the old files on a daily basis for a week, a weekly basis for a month, then monthly for six months. In addition, offsite capability was not needed (well perhaps it really is, but it was never used). Their daily changes amount to about 300MB on the average, or about 2GB per week.

As a consequence, the total volume of data they need to keep to meet their needs is about 100GB (15GB x 6 + 2GB x 5 + 0.3 x 7) = 102.1GB.

The chosen solution was to buy a 120GB hard disk for next to nothing – far less than 1/10th the price of a tape drive and the cassettes to handle the same amount of data, and to have Bacula write to disk files.

The rest of this chapter will explain how to setup Bacula so that it would automatically manage a set of disk files with the minimum sysadmin intervention. The system has been running since 22 January 2004 until today (23 June 2007) with no intervention, with the exception of adding a second 120GB hard disk after a year because their needs grew over that time to more than the 120GB (168GB to be exact). The only other intervention I have made is a periodic (about once a year) Bacula upgrade.

32.3 Overall Design

Getting Bacula to write to disk rather than tape in the simplest case is rather easy, and is documented in the previous chapter. In addition, all the directives discussed here are explained in that chapter. We'll leave it to you to look at the details there. If you haven't read it and are not familiar with Pools, you probably should at least read it once quickly for the ideas before continuing here.

One needs to consider about what happens if we have only a single large Bacula Volume defined on our hard disk. Everything works fine until the Volume fills, then Bacula will ask you to mount a new Volume. This same problem applies to the use of tape Volumes if your tape fills. Being a hard disk and the only one you have, this will be a bit of a problem. It should be obvious that it is better to use a number of smaller Volumes and arrange for Bacula to automatically recycle them so that the disk storage space can be reused. The other problem with a single Volume, is that until version 2.0.0, Bacula did not seek within a disk Volume, so restoring a single file can take more time than one would expect.

As mentioned, the solution is to have multiple Volumes, or files on the disk. To do so, we need to limit the use and thus the size of a single Volume, by time, by number of jobs, or by size. Any of these would work, but we chose to limit the use of a single Volume by putting a single job in each Volume with the exception of Volumes containing Incremental backup where there will be 6 jobs (a week's worth of data) per volume. The details of this will be discussed shortly. This is a single client backup, so if you have multiple clients you will need to multiply those numbers by the number of clients, or use a different system for switching volumes, such as limiting the volume size.

The next problem to resolve is recycling of Volumes. As you noted from above, the requirements are to be able to restore monthly for 6 months, weekly for a month, and daily for a week. So to simplify things, why not do a Full save once a month, a Differential save once a week, and Incremental saves daily. Now since each of these different kinds of saves needs to remain valid for differing periods, the simplest way to do this (and possibly the only) is to have a separate Pool for each backup type.

The decision was to use three Pools: one for Full saves, one for Differential saves, and one for Incremental saves, and each would have a different number of volumes and a different Retention period to accomplish the requirements.

32.3.1 Full Pool

Putting a single Full backup on each Volume, will require six Full save Volumes, and a retention period of six months. The Pool needed to do that is:

```
Pool {
  Name = Full-Pool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 6 months
  Maximum Volume Jobs = 1
  Label Format = Full-
  Maximum Volumes = 9
}
```

Since these are disk Volumes, no space is lost by having separate Volumes for each backup (done once a month in this case). The items to note are the retention period of six months (i.e. they are recycled after six months), that there is one job per volume (Maximum Volume Jobs = 1), the volumes will be labeled Full-0001, ... Full-0006 automatically. One could have labeled these manually from the start, but why not use the features of Bacula.

Six months after the first volume is used, it will be subject to pruning and thus recycling, so with a maximum of 9 volumes, there should always be 3 volumes available (note, they may all be marked used, but they will be marked purged and recycled as needed).

If you have two clients, you would want to set **Maximum Volume Jobs** to 2 instead of one, or set a limit on the size of the Volumes, and possibly increase the maximum number of Volumes.

32.3.2 Differential Pool

For the Differential backup Pool, we choose a retention period of a bit longer than a month and ensure that there is at least one Volume for each of the maximum of five weeks in a month. So the following works:

```
Pool {
  Name = Diff-Pool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 40 days
  Maximum Volume Jobs = 1
  Label Format = Diff-
  Maximum Volumes = 10
}
```

As you can see, the Differential Pool can grow to a maximum of 9 volumes, and the Volumes are retained 40 days and thereafter they can be recycled. Finally there is one job per volume. This, of course, could be tightened up a lot, but the expense here is a few GB which is not too serious.

If a new volume is used every week, after 40 days, one will have used 7 volumes, and there should then always be 3 volumes that can be purged and recycled.

See the discussion above concering the Full pool for how to handle multiple clients.

32.3.3 Incremental Pool

Finally, here is the resource for the Incremental Pool:

```
Pool {
  Name = Inc-Pool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 20 days
  Maximum Volume Jobs = 6
  Label Format = Inc-
  Maximum Volumes = 7
}
```

We keep the data for 20 days rather than just a week as the needs require. To reduce the proliferation of volume names, we keep a week's worth of data (6 incremental backups) in each Volume. In practice, the retention period should be set to just a bit more than a week and keep only two or three volumes instead of five. Again, the lost is very little and as the system reaches the full steady state, we can adjust these values so that the total disk usage doesn't exceed the disk capacity.

If you have two clients, the simplest thing to do is to increase the maximum volume jobs from 6 to 12. As mentioned above, it is also possible limit the size of the volumes. However, in that case, you will need to have a better idea of the volume or add sufficient volumes to the pool so

that you will be assured that in the next cycle (after 20 days) there is at least one volume that is pruned and can be recycled.

32.4 The Actual Conf Files

The following example shows you the actual files used, with only a few minor modifications to simplify things.

The Director's configuration file is as follows:

```
Director {          # define myself
  Name = bacula-dir
  DIRport = 9101
  QueryFile = "/opt/bacula/bin/query.sql"
  WorkingDirectory = "/opt/bacula/working"
  PidDirectory = "/opt/bacula/working"
  Maximum Concurrent Jobs = 1
  Password = " *** CHANGE ME ***"
  Messages = Standard
}
#   By default, this job will back up to disk in /tmp
Job {
  Name = client
  Type = Backup
  Client = client-fd
  FileSet = "Full Set"
  Schedule = "WeeklyCycle"
  Storage = File
  Messages = Standard
  Pool = Default
  Full Backup Pool = Full-Pool
  Incremental Backup Pool = Inc-Pool
  Differential Backup Pool = Diff-Pool
  Write Bootstrap = "/opt/bacula/working/client.bsr"
  Priority = 10
}

# Backup the catalog database (after the nightly save)
Job {
  Name = "BackupCatalog"
  Type = Backup
  Client = client-fd
  FileSet="Catalog"
  Schedule = "WeeklyCycleAfterBackup"
  Storage = File
  Messages = Standard
  Pool = Default
  # This creates an ASCII copy of the catalog
  # WARNING!!! Passing the password via the command line is insecure.
  # see comments in make_catalog_backup.pl for details.
  RunBeforeJob = "/opt/bacula/bin/make_catalog_backup.pl MyCatalog"
  # This deletes the copy of the catalog
  RunAfterJob  = "/opt/bacula/bin/delete_catalog_backup"
  Write Bootstrap = "/opt/bacula/working/BackupCatalog.bsr"
  Priority = 11                   # run after main backup
}

# Standard Restore template, to be changed by Console program
Job {
  Name = "RestoreFiles"
  Type = Restore
  Client = havana-fd
  FileSet="Full Set"
  Storage = File
  Messages = Standard
  Pool = Default
  Where = /tmp/bacula-restores
}
```

```
# List of files to be backed up
FileSet {
  Name = "Full Set"
  Include = { Options { signature=SHA1; compression=GZIP9 }
    File = /
    File = /usr
    File = /home
    File = /boot
    File = /var
    File = /opt
  }
  Exclude = {
    File = /proc
    File = /tmp
    File = /.journal
    File = /.fsck
    ...
  }
}
Schedule {
  Name = "WeeklyCycle"
  Run = Level=Full 1st sun at 2:05
  Run = Level=Differential 2nd-5th sun at 2:05
  Run = Level=Incremental mon-sat at 2:05
}

# This schedule does the catalog. It starts after the WeeklyCycle
Schedule {
  Name = "WeeklyCycleAfterBackup"
  Run = Level=Full sun-sat at 2:10
}

# This is the backup of the catalog
FileSet {
  Name = "Catalog"
  Include { Options { signature=MD5 }
    File = /opt/bacula/working/bacula.sql
  }
}

Client {
  Name = client-fd
  Address = client
  FDPort = 9102
  Catalog = MyCatalog
  Password = " *** CHANGE ME ***"
  AutoPrune = yes      # Prune expired Jobs/Files
  Job Retention = 6 months
  File Retention = 60 days
}

Storage {
  Name = File
  Address = localhost
  SDPort = 9103
  Password = " *** CHANGE ME ***"
  Device = FileStorage
  Media Type = File
}

Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}

Pool {
  Name = Full-Pool
  Pool Type = Backup
  Recycle = yes           # automatically recycle Volumes
  AutoPrune = yes         # Prune expired volumes
  Volume Retention = 6 months
  Maximum Volume Jobs = 1
  Label Format = Full-
```

```
  Maximum Volumes = 9
}

Pool {
  Name = Inc-Pool
  Pool Type = Backup
  Recycle = yes            # automatically recycle Volumes
  AutoPrune = yes          # Prune expired volumes
  Volume Retention = 20 days
  Maximum Volume Jobs = 6
  Label Format = Inc-
  Maximum Volumes = 7
}

Pool {
  Name = Diff-Pool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 40 days
  Maximum Volume Jobs = 1
  Label Format = Diff-
  Maximum Volumes = 10
}

Messages {
  Name = Standard
  mailcommand = "bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
      -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
      -s \"Bacula: Intervention needed for %j\" %r"
  mail = root@domain.com = all, !skipped
  operator = root@domain.com = mount
  console = all, !skipped, !saved
  append = "/opt/bacula/bin/log" = all, !skipped
}
```

and the Storage daemon's configuration file is:

```
Storage {               # definition of myself
  Name = bacula-sd
  SDPort = 9103       # Director's port
  WorkingDirectory = "/opt/bacula/working"
  Pid Directory = "/opt/bacula/working"
}
Director {
  Name = bacula-dir
  Password = " *** CHANGE ME ***"
}
Device {
  Name = FileStorage
  Media Type = File
  Archive Device = /files/bacula
  LabelMedia = yes;    # lets Bacula label unlabeled media
  Random Access = Yes;
  AutomaticMount = yes;   # when device opened, read it
  RemovableMedia = no;
  AlwaysOpen = no;
}
Messages {
  Name = Standard
  director = bacula-dir = all
}
```

Chapter 33

Migration and Copy

The term Migration, as used in the context of Bacula, means moving data from one Volume to another. In particular it refers to a Job (similar to a backup job) that reads data that was previously backed up to a Volume and writes it to another Volume. As part of this process, the File catalog records associated with the first backup job are purged. In other words, Migration moves Bacula Job data from one Volume to another by reading the Job data from the Volume it is stored on, writing it to a different Volume in a different Pool, and then purging the database records for the first Job.

The Copy process is essentially identical to the Migration feature with the exception that the Job that is copied is left unchanged. This essentially creates two identical copies of the same backup. However, the copy is treated as a copy rather than a backup job, and hence is not directly available for restore. If bacula founds a copy when a job record is purged (deleted) from the catalog, it will promote the copy as *real* backup and will make it available for automatic restore. Note: in the text below, to simplify it, we usually speak of a migration job. This, in fact, means either a migration job or a copy job.

The Copy and the Migration jobs run without using the File daemon by copying the data from the old backup Volume to a different Volume in a different Pool. It is not possible to run commands on the defined Client via a RunScript from within the Migration or Copy Job.

The selection process for which Job or Jobs are migrated can be based on quite a number of different criteria such as:

- a single previous Job
- a Volume
- a Client
- a regular expression matching a Job, Volume, or Client name
- the time a Job has been on a Volume
- high and low water marks (usage or occupation) of a Pool
- Volume size

The details of these selection criteria will be defined below.

To run a Migration job, you must first define a Job resource very similar to a Backup Job but with **Type = Migrate** instead of **Type = Backup**. One of the key points to remember is that the Pool that is specified for the migration job is the only pool from which jobs will be migrated, with one exception noted below. In addition, the Pool to which the selected Job or Jobs will be migrated is defined by the **Next Pool = ...** in the Pool resource specified for the Migration Job.

Bacula permits Pools to contain Volumes with different Media Types. However, when doing migration, this is a very undesirable condition. For migration to work properly, you should use Pools containing only Volumes of the same Media Type for all migration jobs.

The migration job normally is either manually started or starts from a Schedule much like a backup job. It searches for a previous backup Job or Jobs that match the parameters you have specified in the migration Job resource, primarily a **Selection Type** (detailed a bit later). Then for each previous backup JobId found, the Migration Job will run a new Job which copies the old Job data from the previous Volume to a new Volume in the Migration Pool. It is possible that no prior Jobs are found for migration, in which case, the Migration job will simply terminate having done nothing, but normally at a minimum, three jobs are involved during a migration:

- The currently running Migration control Job. This is only a control job for starting the migration child jobs.
- The previous Backup Job (already run). The File records for this Job are purged if the Migration job successfully terminates. The original data remains on the Volume until it is recycled and rewritten.
- A new Migration Backup Job that moves the data from the previous Backup job to the new Volume. If you subsequently do a restore, the data will be read from this Job.

If the Migration control job finds a number of JobIds to migrate (e.g. it is asked to migrate one or more Volumes), it will start one new migration backup job for each JobId found on the specified Volumes. Please note that Migration doesn't scale too well since Migrations are done on a Job by Job basis. This if you select a very large volume or a number of volumes for migration, you may have a large number of Jobs that start. Because each job must read the same Volume, they will run consecutively (not simultaneously).

33.1 Migration and Copy Job Resource Directives

The following directives can appear in a Director's Job resource, and they are used to define a Migration job.

Pool = <Pool-name> The Pool specified in the Migration control Job is not a new directive for the Job resource, but it is particularly important because it determines what Pool will be examined for finding JobIds to migrate. The exception to this is when **Selection Type = SQLQuery**, and although a Pool directive must still be specified, no Pool is used, unless you specifically include it in the SQL query. Note, in any case, the Pool resource defined by the Pool directive must contain a **Next Pool = ...** directive to define the Pool to which the data will be migrated.

Type = Migrate **Migrate** is a new type that defines the job that is run as being a Migration Job. A Migration Job is a sort of control job and does not have any Files associated with it, and in that sense they are more or less like an Admin job. Migration jobs simply check to see if there is anything to Migrate then possibly start and control new Backup jobs to migrate the data from the specified Pool to another Pool. Note, any original JobId that is migrated will be marked as having been migrated, and the original JobId can nolonger be used for restores; all restores will be done from the new migrated Job.

Type = Copy **Copy** is a new type that defines the job that is run as being a Copy Job. A Copy Job is a sort of control job and does not have any Files associated with it, and in that sense they are more or less like an Admin job. Copy jobs simply check to see if there is anything to Copy then possibly start and control new Backup jobs to copy the data from the specified Pool to another Pool. Note that when a copy is made, the original JobIds are left unchanged. The new copies can not be used for restoration unless you specifically choose them by JobId. Also you subsequently delete a JobId that has a copy, the copy

will be automatically upgraded to a Backup rather than a Copy, and it will subsequently be used for restoration.

Once again, a Copy Job cannot be used to restore unless you explicitly specify the Copy JobId during the restore command. If the original backup Job is deleted and there is a Copy of that backup Job, the Copy JobIds will be changed to be backup Jobs that can then be restored.

Selection Type = <Selection-type-keyword> The <Selection-type-keyword> determines how the migration job will go about selecting what JobIds to migrate. In most cases, it is used in conjunction with a **Selection Pattern** to give you fine control over exactly what JobIds are selected. The possible values for <Selection-type-keyword> are:

SmallestVolume This selection keyword selects the volume with the fewest bytes from the Pool to be migrated. The Pool to be migrated is the Pool defined in the Migration Job resource. The migration control job will then start and run one migration backup job for each of the Jobs found on this Volume. The Selection Pattern, if specified, is not used.

OldestVolume This selection keyword selects the volume with the oldest last write time in the Pool to be migrated. The Pool to be migrated is the Pool defined in the Migration Job resource. The migration control job will then start and run one migration backup job for each of the Jobs found on this Volume. The Selection Pattern, if specified, is not used.

Client The Client selection type, first selects all the Clients that have been backed up in the Pool specified by the Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Client names, giving a filtered Client name list. All jobs that were backed up for those filtered (regexed) Clients will be migrated. The migration control job will then start and run one migration backup job for each of the JobIds found for those filtered Clients.

Volume The Volume selection type, first selects all the Volumes that have been backed up in the Pool specified by the Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Volume names, giving a filtered Volume list. All JobIds that were backed up for those filtered (regexed) Volumes will be migrated. The migration control job will then start and run one migration backup job for each of the JobIds found on those filtered Volumes.

Jobs on Volumes will be considered for Migration only if the Volume is marked, Full, Used, or Error. Volumes that are still marked Append will not be considered for migration. This prevents Bacula from attempting to read the Volume at the same time it is writing it. It also reduces other deadlock situations, as well as avoids the problem that you migrate a Volume and later find new files appended to that Volume.

Job The Job selection type, first selects all the Jobs (as defined on the **Name** directive in a Job resource) that have been backed up in the Pool specified by the Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Job names, giving a filtered Job name list. All JobIds that were run for those filtered (regexed) Job names will be migrated. Note, for a given Job named, they can be many jobs (JobIds) that ran. The migration control job will then start and run one migration backup job for each of the Jobs found.

SQLQuery The SQLQuery selection type, used the **Selection Pattern** as an SQL query to obtain the JobIds to be migrated. The Selection Pattern must be a valid SELECT SQL statement for your SQL engine, and it must return the JobId as the first field of the SELECT.

PoolOccupancy This selection type will cause the Migration job to compute the total size of the specified pool for all Media Types combined. If it exceeds the **Migration High Bytes** defined in the Pool, the Migration job will migrate all JobIds beginning with the oldest Volume in the pool (determined by Last Write time) until the Pool bytes drop below the **Migration Low Bytes** defined in the Pool. This calculation should be consider rather approximate because it is made once by the Migration job before migration is begun, and thus does not take into account additional data written into the Pool during the migration. In addition, the calculation of the total Pool byte size

is based on the Volume bytes saved in the Volume (Media) database entries. The bytes calculate for Migration is based on the value stored in the Job records of the Jobs to be migrated. These do not include the Storage daemon overhead as is in the total Pool size. As a consequence, normally, the migration will migrate more bytes than strictly necessary.

PoolTime The PoolTime selection type will cause the Migration job to look at the time each JobId has been in the Pool since the job ended. All Jobs in the Pool longer than the time specified on **Migration Time** directive in the Pool resource will be migrated.

PoolUncopiedJobs This selection which copies all jobs from a pool to an other pool which were not copied before is available only for copy Jobs.

Selection Pattern = <Quoted-string> The Selection Patterns permitted for each Selection-type-keyword are described above.

For the OldestVolume and SmallestVolume, this Selection pattern is not used (ignored).

For the Client, Volume, and Job keywords, this pattern must be a valid regular expression that will filter the appropriate item names found in the Pool.

For the SQLQuery keyword, this pattern must be a valid SELECT SQL statement that returns JobIds.

Purge Migration Job = <yes|no> This directive may be added to the Migration Job definition in the Director configuration file to purge the job migrated at the end of a migration.

33.2 Migration Pool Resource Directives

The following directives can appear in a Director's Pool resource, and they are used to define a Migration job.

Migration Time = <time-specification> If a PoolTime migration is done, the time specified here in seconds (time modifiers are permitted – e.g. hours, ...) will be used. If the previous Backup Job or Jobs selected have been in the Pool longer than the specified PoolTime, then they will be migrated.

Migration High Bytes = <byte-specification> This directive specifies the number of bytes in the Pool which will trigger a migration if a **PoolOccupancy** migration selection type has been specified. The fact that the Pool usage goes above this level does not automatically trigger a migration job. However, if a migration job runs and has the PoolOccupancy selection type set, the Migration High Bytes will be applied. Bacula does not currently restrict a pool to have only a single Media Type, so you must keep in mind that if you mix Media Types in a Pool, the results may not be what you want, as the Pool count of all bytes will be for all Media Types combined.

Migration Low Bytes = <byte-specification> This directive specifies the number of bytes in the Pool which will stop a migration if a **PoolOccupancy** migration selection type has been specified and triggered by more than Migration High Bytes being in the pool. In other words, once a migration job is started with **PoolOccupancy** migration selection and it determines that there are more than Migration High Bytes, the migration job will continue to run jobs until the number of bytes in the Pool drop to or below Migration Low Bytes.

Next Pool = <pool-specification> The Next Pool directive specifies the pool to which Jobs will be migrated. This directive is required to define the Pool into which the data will be migrated. Without this directive, the migration job will terminate in error.

The Next Pool directive may also be specified in the Job resource or on a Run directive in the Schedule resource. Any Next Pool directive in the Job resource will take precedence over the Pool definition, and any Next Pool specification on the Run directive in a Schedule resource will take ultimate precedence.

Storage = <storage-specification> The Storage directive specifies what Storage resource will be used for all Jobs that use this Pool. It takes precedence over any other Storage specifications that may have been given such as in the Schedule Run directive, or in the Job resource. We highly recommend that you define the Storage resource to be used in the Pool rather than elsewhere (job, schedule run, ...).

33.3 Important Migration Considerations

- Each Pool into which you migrate Jobs or Volumes **must** contain Volumes of only one Media Type.
- Migration takes place on a JobId by JobId basis. That is each JobId is migrated in its entirety and independently of other JobIds. Once the Job is migrated, it will be on the new medium in the new Pool, but for the most part, aside from having a new JobId, it will appear with all the same characteristics of the original job (start, end time, ...). The column RealEndTime in the catalog Job table will contain the time and date that the Migration terminated, and by comparing it with the EndTime column you can tell whether or not the job was migrated. The original job is purged of its File records, and its Type field is changed from "B" to "M" to indicate that the job was migrated.
- As noted above, for the Migration High Bytes, the calculation of the bytes to migrate is somewhat approximate.
- If you keep Volumes of different Media Types in the same Pool, it is not clear how well migration will work. We recommend only one Media Type per pool.
- It is possible to get into a resource deadlock where Bacula does not find enough drives to simultaneously read and write all the Volumes needed to do Migrations. For the moment, you must take care as all the resource deadlock algorithms are not yet implemented. You can also use the **Maximum Concurrent Read Jobs** directive to control the drive assignment behavior to reduce resource deadlocks. The MaximumConcurrentReadJobs should have a maximum of one less than one half the number of drives that are available for reading and writing. Doing so, will assure that for each read drive there is one that can write so that the Migration jobs will not stall due to a deadlock.
- Migration is done only when you run a Migration job. If you set a Migration High Bytes and that number of bytes is exceeded in the Pool no migration job will automatically start. You must schedule the migration jobs, and they must run for any migration to take place.
- If you migrate a number of Volumes, a very large number of Migration jobs may start.
- Figuring out what jobs will actually be migrated can be a bit complicated due to the flexibility provided by the regex patterns and the number of different options. Turning on a debug level of 100 or more will provide a limited amount of debug information about the migration selection process.
- Bacula currently does only minimal Storage conflict resolution, so you must take care to ensure that you don't try to read and write to the same device or Bacula may block waiting to reserve a drive that it will never find. In general, ensure that all your migration pools contain only one Media Type, and that you always migrate to pools with different Media Types.
- The **Next Pool = ...** directive must be defined in the Pool referenced in the Migration Job to define the Pool into which the data will be migrated.
- Pay particular attention to the fact that data is migrated on a Job by Job basis, and for any particular tape Volume, only one Job can read that Volume at a time (no simultaneous read), so migration jobs that all reference the same Volume will run sequentially. This can be a potential bottle neck and does not scale very well to large numbers of jobs. For disk Volumes, multiple simultaneous Jobs can read the same Volume at the same time, so the above restriction does not apply.
- Only migration of Selection Types of Job and Volume have been carefully tested. All the other migration methods (time, occupancy, smallest, oldest, ...) need additional testing.

- Plugins are not compatible with Copy/Migration/VirtualFull jobs by default. Please contact Bacula Systems support team to know if your strategy will be possible with your setup.
- Windows backups using VSS writers (normally plugins) are not compatible with Migration and Copy. The reason is that VSS writers require that certain **XML!** files generated at the end of the Backup, but they must be presented to the VSS writer at the beginning of a restore Job. This is very difficult for a backup program such as Bacula. Just the same Bacula accomplishes this by saving the VSS writer's **XML!** files in Restore Objects in the catalog. However, during Migration and Copy Jobs these Restore Objects are not currently copied to the new Job.

33.4 Example Migration Jobs

When you specify a Migration Job, you must specify all the standard directives as for a Job. However, certain such as the Level, Client, and FileSet, though they must be defined, are ignored by the Migration job because the values from the original job used instead.

As an example, suppose you have the following Job that you run every night. To note: there is no Storage directive in the Job resource; there is a Storage directive in each of the Pool resources; the Pool to be migrated (File) contains a Next Pool directive that defines the output Pool (where the data is written by the migration job).

```
# Define the backup Job
Job {
  Name = "NightlySave"
  Type = Backup
  Level = Incremental                 # default
  Client=rufus-fd
  FileSet="Full Set"
  Schedule = "WeeklyCycle"
  Messages = Standard
  Pool = Default
}

# Default pool definition
Pool {
  Name = Default
  Pool Type = Backup
  AutoPrune = yes
  Recycle = yes
  Next Pool = Tape
  Storage = File
  LabelFormat = "File"
}

# Tape pool definition
Pool {
  Name = Tape
  Pool Type = Backup
  AutoPrune = yes
  Recycle = yes
  Storage = DLTDrive
}

# Definition of File storage device
Storage {
  Name = File
  Address = rufus
  Password = "ccV3lVTsQRsdIUGyab0N4sMDavui2hOBkmpBU0aQKOr9"
  Device = "File"          # same as Device in Storage daemon
  Media Type = File        # same as MediaType in Storage daemon
}

# Definition of DLT tape storage device
Storage {
```

```
  Name = DLTDrive
  Address = rufus
  Password = "ccV3lVTsQRsdIUGyab0N4sMDavui2hOBkmpBUOaQKOr9"
  Device = "HP DLT 80"       # same as Device in Storage daemon
  Media Type = DLT8000       # same as MediaType in Storage daemon
}
```

Where we have included only the essential information – i.e. the Director, FileSet, Catalog, Client, Schedule, and Messages resources are omitted.

As you can see, by running the NightlySave Job, the data will be backed up to File storage using the Default pool to specify the Storage as File.

Now, if we add the following Job resource to this conf file.

```
Job {
  Name = "migrate-volume"
  Type = Migrate
  Level = Full
  Client = rufus-fd
  FileSet = "Full Set"
  Messages = Standard
  Pool = Default
  Maximum Concurrent Jobs = 4
  Selection Type = Volume
  Selection Pattern = "File"
}
```

and then run the job named **migrate-volume**, all volumes in the Pool named Default (as specified in the migrate-volume Job that match the regular expression pattern **File** will be migrated to tape storage DLTDrive because the **Next Pool** in the Default Pool specifies that Migrations should go to the pool named **Tape**, which uses Storage **DLTDrive**.

If instead, we use a Job resource as follows:

```
Job {
  Name = "migrate"
  Type = Migrate
  Level = Full
  Client = rufus-fd
  FileSet="Full Set"
  Messages = Standard
  Pool = Default
  Maximum Concurrent Jobs = 4
  Selection Type = Job
  Selection Pattern = ".*Save"
}
```

All jobs ending with the name Save will be migrated from the File Default to the Tape Pool, or from File storage to Tape storage.

33.5 Virtual Backup Consolidation

When the Job Level is set to VirtualFull, it permits you to consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

Bacula's virtual backup feature is often called Synthetic Backup or Consolidation in other backup products.

In some respects the Virtual Backup feature works similar to a Migration job, in that Bacula normally reads the data from the pool specified in the Job resource, and writes it to the **Next Pool** specified in the Job resource. Note, this means that usually the output from the Virtual Backup is written into a different pool from where your prior backups are saved. Doing it this way guarantees that you will not get a deadlock situation attempting to read and write to the same volume in the Storage daemon. If you then want to do subsequent backups, you may need to move the Virtual Full Volume back to your normal backup pool. Alternatively, you can set your **Next Pool** to point to the current pool. This will cause Bacula to read and write to Volumes in the current pool. In general, this will work, because Bacula will not allow reading and writing on the same Volume. In any case, once a VirtualFull has been created, and a restore is done involving the most current Full, it will read the Volume or Volumes by the VirtualFull regardless of in which Pool the Volume is found.

A typical Job resource definition might look like the following:

```
Job {
  Name = "MyBackup"
  Type = Backup
  Client=localhost-fd
  FileSet = "Full Set"
  Storage = File
  Messages = Standard
  Pool = Default
  SpoolData = yes
}

# Default pool definition
Pool {
  Name = Default
  Pool Type = Backup
  Volume Retention = 365d  # one year
  NextPool = Full
  Storage = File
}

Pool {
  Name = Full
  Pool Type = Backup
  Volume Retention = 365d  # one year
  Storage = DiskChanger
}

# Definition of file storage device
Storage {
  Name = File
  Address = localhost
  Password = "xxx"
  Device = FileStorage
  Media Type = File
  Maximum Concurrent Jobs = 5
}

# Definition of DDS Virtual tape disk storage device
Storage {
  Name = DiskChanger
  Address = localhost  # N.B. Use a fully qualified name here
  Password = "yyy"
  Device = DiskChanger
  Media Type = DiskChangerMedia
  Maximum Concurrent Jobs = 4
  Autochanger = yes
}
```

Then in bconsole or via a Run schedule, you would run the job as:

```
run job=MyBackup level=Full
run job=MyBackup level=Incremental
run job=MyBackup level=Differential
```

```
run job=MyBackup level=Incremental
run job=MyBackup level=Incremental
```

So providing there were changes between each of those jobs, you would end up with a Full backup, a Differential, which includes the first Incremental backup, then two Incremental backups. All the above jobs would be written to the **Default** pool.

To consolidate those backups into a new Full backup, you would run the following:

```
run job=MyBackup level=VirtualFull
```

And it would produce a new Full backup without using the client, and the output would be written to the **Full** Pool which uses the Diskchanger Storage.

If the Virtual Full is run, and there are no prior Jobs, the Virtual Full will fail with an error.

Note, the Start and End time of the Virtual Full backup is set to the values for the last job included in the Virtual Full (in the above example, it is an Increment). This is so that if another incremental is done, which will be based on the Virtual Full, it will backup all files from the last Job included in the Virtual Full rather than from the time the Virtual Full was actually run.

33.5.1 Manually Specify Jobs to Consolidate

By default Bacula is selecting jobs automatically, however, you may want to choose any point in time to create the Virtual backup.

For example, if you have the following Jobs in your catalog:

```
+-------+---------+-------+----------+----------+-----------+
| JobId | Name    | Level | JobFiles | JobBytes | JobStatus |
+-------+---------+-------+----------+----------+-----------+
| 1     | Vbackup | F     | 1754     | 50118554 | T         |
| 2     | Vbackup | I     | 1        | 4        | T         |
| 3     | Vbackup | I     | 1        | 4        | T         |
| 4     | Vbackup | D     | 2        | 8        | T         |
| 5     | Vbackup | I     | 1        | 6        | T         |
| 6     | Vbackup | I     | 10       | 60       | T         |
| 7     | Vbackup | I     | 11       | 65       | T         |
| 8     | Save    | F     | 1758     | 50118564 | T         |
+-------+---------+-------+----------+----------+-----------+
```

If you want to consolidate only the first 3 jobs and create a virtual backup equivalent to Job 1 + Job 2 + Job 3, you will use `jobid=3` in the `run` command, then Bacula will select the previous Full backup, the previous Differential (if any) and all subsequent Incremental jobs.

```
run job=Vbackup jobid=3 level=VirtualFull
```

If you want to consolidate a specific job list, you must specify the exact list of jobs to merge in the `run` command line. For example, to consolidate the last Differential and all subsequent Incremental, you will use `jobid=4,5,6,7` or `jobid=4-7` in the `run` command line. As one of the Job in the list is a Differential backup, Bacula will set the new job level to Differential. If the list is composed only with Incremental jobs, the new job will have a level set to Incremental.

```
run job=Vbackup jobid=4-7 level=VirtualFull
```

When using this feature, Bacula will automatically discard jobs that are not related to the current Job. For example, specifying `jobid=7,8`, Bacula will discard the jobid 8.

If you know what you are doing and still want to consolidate jobs that have different names (so probably different clients, filesets, etc. . .), you must use `alljobid=` keyword instead of `jobid=`.

```
run job=Vbackup alljobid=1-3,6-8 level=VirtualFull
```

Chapter 34

File Deduplication using Base Jobs

A base job is sort of like a Full save except that you will want the FileSet to contain only files that are unlikely to change in the future (i.e. a snapshot of most of your system after installing it). After the base job has been run, when you are doing a Full save, you specify one or more Base jobs to be used. All files that have been backed up in the Base job/jobs but not modified will then be excluded from the backup. During a restore, the Base jobs will be automatically pulled in where necessary.

This can be a very nice optimization for your backups. Basically, imagine that you have 100 nearly identical Windows or Linux machines containing the OS and user files. Now for the OS part, a Base job will be backed up once, and rather than making 100 copies of the OS, there will be only one. If one or more of the systems have some files updated, no problem, they will be automatically saved and restored.

A new Job directive `Base=Jobx, Joby...` permits you to specify the list of jobs that will be used during a Full backup as base.

```
Job {
   Name = BackupLinux
   Level= Base
   ...
}

Job {
   Name = BackupZog4
   Base = BackupZog4, BackupLinux
   Accurate = yes
   ...
}
```

In this example, the job `BackupZog4` will use the most recent version of all files contained in `BackupZog4` and `BackupLinux` jobs. Base jobs should have run with `level=Base` to be used.

By default, Bacula will compare permissions bits, user and group fields, modification time, size and the checksum of the file to choose between the current backup and the BaseJob file list. You can change this behavior with the `BaseJob` FileSet option. This option works like the `verify=` one, that is described in the FileSet chapter.

```
FileSet {
  Name = Full
  Include = {
    Options {
       BaseJob  = pmugcs5
       Accurate = mcs
       Verify   = pin5
    }
```

```
    File = /
  }
}
```

Important note : The current implementation doesn't permit to scan a Volume with the bscan facility. The result does not permit the restore of files. It is therefore recommended to not prune File or Job records with Basejobs in use.

Added in version 8.0.5, the new "M" option letter for the Accurate directive in the FileSet Options block, which allows comparing the modification time and/or creation time against the last backup timestamp. This is in contrast to the existing options letters "m" and/or "c", mtime and ctime, which are checked against the stored catalog values, that could vary accross different machines when using the BaseJob feature.

The advantage of the new "M" option letter for Jobs that refer to BaseJobs is that it will backup files based on the last backup time, which is more useful, because the mtime/ctime timestamps may differ on various Clients, causing unnecessary files to be backed up.

```
Job {
    Name = USR
    Level = Base
    FileSet = BaseFS
    ...
}

Job {
    Name = Full
    FileSet = FullFS
    Base = USR
    ...
}

FileSet {
    Name = BaseFS
    Include {
      Options {
        Signature = MD5
      }
      File = /usr
    }
}

FileSet {
    Name = FullFS
    Include {
      Options {
        Accurate = Ms      # check for mtime/ctime and Size
        Signature = MD5
      }
      File = /home
      File = /usr
    }
}
```

Base Jobs are not compatible with Copy or Migration Jobs.

Chapter 35

Backup Strategies

Although Recycling and Backing Up to Disk Volume have been discussed in previous chapters, this chapter is meant to give you an overall view of possible backup strategies and to explain their advantages and disadvantages.

35.1 Simple One Tape Backup

Probably the simplest strategy is to back everything up to a single tape and insert a new (or recycled) tape when it fills and Bacula requests a new one.

35.1.1 Advantages

- The operator intervenes only when a tape change is needed. (once a month at my site).
- There is little chance of operator error because the tape is not changed daily.
- A minimum number of tapes will be needed for a full restore. Typically the best case will be one tape and worst two.
- You can easily arrange for the Full backup to occur a different night of the month for each system, thus load balancing and shortening the backup time.

35.1.2 Disadvantages

- If your site burns down, you will lose your current backups, and in my case about a month of data.
- After a tape fills and you have put in a blank tape, the backup will continue, and this will generally happen during working hours.

35.1.3 Practical Details

This system is very simple. When the tape fills and Bacula requests a new tape, you `unmount` the tape from the Console program, insert a new tape and `label` it. In most cases after the label, Bacula will automatically mount the tape and resume the backup. Otherwise, you simply `mount` the tape.

Using this strategy, one typically does a Full backup once a week followed by daily Incremental backups. To minimize the amount of data written to the tape, one can do a Full backup once

a month on the first Sunday of the month, a Differential backup on the 2nd-5th Sunday of the month, and incremental backups the rest of the week.

35.2 Manually Changing Tapes

If you use the strategy presented above, Bacula will ask you to change the tape, and you will `unmount` it and then remount it when you have inserted the new tape.

If you do not wish to interact with Bacula to change each tape, there are several ways to get Bacula to release the tape:

- In your Storage daemon's Device resource, set **AlwaysOpen = no** In this case, Bacula will release the tape after every job. If you run several jobs, the tape will be rewound and repositioned to the end at the beginning of every job. This is not very efficient, but does let you change the tape whenever you want.
- Use a **RunAfterJob** statement to run a script after your last job. This could also be an **Admin** job that runs after all your backup jobs. The script could be something like:

  ```
  #!/bin/sh
  /opt/bacula/bin/bconsole -c /opt/bacula/bin/bconsole.conf <<END_OF_DATA
  release storage=your-storage-name
  END_OF_DATA
  ```

  In this example, you would have **AlwaysOpen=yes**, but the `release` command would tell Bacula to rewind the tape and on the next job assume the tape has changed. This strategy may not work on some systems, or on autochangers because Bacula will still keep the drive open.
- The final strategy is similar to the previous case except that you would use the `unmount` command to force Bacula to release the drive. Then you would eject the tape, and remount it as follows:

  ```
  #!/bin/sh
  /opt/bacula/bin/bconsole -c /opt/bacula/bin/bconsole.conf <<END_OF_DATA
  unmount storage=your-storage-name
  END_OF_DATA
  # the following is a shell command
  mt eject
  /opt/bacula/bin/bconsole -c /opt/bacula/bin/bconsole.conf <<END_OF_DATA
  mount storage=your-storage-name
  END_OF_DATA
  ```

35.3 Daily Tape Rotation

This scheme is quite different from the one mentioned above in that a Full backup is done to a different tape every day of the week. Generally, the backup will cycle continuously through five or six tapes each week. Variations are to use a different tape each Friday, and possibly at the beginning of the month. Thus if backups are done Monday through Friday only, you need only five tapes, and by having two Friday tapes, you need a total of six tapes. Many sites run this way, or using modifications of it based on two week cycles or longer.

35.3.1 Advantages

- All the data is stored on a single tape, so recoveries are simple and faster.
- Assuming the previous day's tape is taken offsite each day, a maximum of one days data will be lost if the site burns down.

35.3.2 Disadvantages

- The tape must be changed every day requiring a lot of operator intervention.
- More errors will occur because of human mistakes.
- If the wrong tape is inadvertently mounted, the Backup for that day will not occur exposing the system to data loss.
- There is much more movement of the tape each day (rewinds) leading to shorter tape drive life time.
- Initial setup of Bacula to run in this mode is more complicated than the Single tape system described above.
- Depending on the number of systems you have and their data capacity, it may not be possible to do a Full backup every night for time reasons or reasons of tape capacity.

35.3.3 Practical Details

The simplest way to "force" Bacula to use a different tape each day is to define a different Pool for each day of the the week a backup is done. In addition, you will need to specify appropriate Job and File retention periods so that Bacula will relabel and overwrite the tape each week rather than appending to it. Nic Bellamy has supplied an actual working model of this which we include here.

What is important is to create a different Pool for each day of the week, and on the **run** statement in the Schedule, to specify which Pool is to be used. He has one Schedule that accomplishes this, and a second Schedule that does the same thing for the Catalog backup run each day after the main backup (Priorities were not available when this script was written). In addition, he uses a **Max Start Delay** of 22 hours so that if the wrong tape is premounted by the operator, the job will be automatically canceled, and the backup cycle will re-synchronize the next day. He has named his Friday Pool **WeeklyPool** because in that Pool, he wishes to have several tapes to be able to restore to a time older than one week.

And finally, in his Storage daemon's Device resource, he has **Automatic Mount = yes** and **Always Open = No**. This is necessary for the tape ejection to work in his **end_of_backup.sh** script below.

For example, his `bacula-dir.conf` file looks like the following:

```
# /etc/bacula/bacula-dir.conf
#
# Bacula Director Configuration file
#
Director {
  Name = ServerName
  DIRport = 9101
  QueryFile = "/etc/bacula/query.sql"
  WorkingDirectory = "/opt/bacula/working"
  PidDirectory = "/opt/bacula/working"
  SubSysDirectory = "/opt/bacula/working"
  Maximum Concurrent Jobs = 1
  Password = "console-pass"
  Messages = Standard
}
#
# Define the main nightly save backup job
#
Job {
  Name = "NightlySave"
  Type = Backup
  Client = ServerName
  FileSet = "Full Set"
  Schedule = "WeeklyCycle"
  Storage = Tape
```

```
  Messages = Standard
  Pool = Default
  Write Bootstrap = "/opt/bacula/bsr/NightlySave.bsr"
  Max Start Delay = 22h
}
# Backup the catalog database (after the nightly save)
Job {
  Name = "BackupCatalog"
  Type = Backup
  Client = ServerName
  FileSet = "Catalog"
  Schedule = "WeeklyCycleAfterBackup"
  Storage = Tape
  Messages = Standard
  Pool = Default
  # This creates an ASCII copy of the catalog
  # WARNING!!! Passing the password via the command line is insecure.
  # see comments in make_catalog_backup.pl for details.
  RunBeforeJob = "/opt/bacula/make_catalog_backup.pl MyCatalog"
  # This deletes the copy of the catalog, and ejects the tape
  RunAfterJob  = "/etc/bacula/end_of_backup.sh"
  Write Bootstrap = "/opt/bacula/bsr/BackupCatalog.bsr"
  Max Start Delay = 22h
}
# Standard Restore template, changed by Console program
Job {
  Name = "RestoreFiles"
  Type = Restore
  Client = ServerName
  FileSet = "Full Set"
  Storage = Tape
  Messages = Standard
  Pool = Default
  Where = /tmp/bacula-restores
}
# List of files to be backed up
FileSet {
  Name = "Full Set"
  Include {
    Options {
      signature=MD5
    }
    File = /
    File = /data
  }
  Exclude = {
    File = /proc
    File = /tmp
    File = /.journal
  }
}
#
# When to do the backups
#
Schedule {
  Name = "WeeklyCycle"
  Run = Level=Full Pool=MondayPool Monday at 8:00pm
  Run = Level=Full Pool=TuesdayPool Tuesday at 8:00pm
  Run = Level=Full Pool=WednesdayPool Wednesday at 8:00pm
  Run = Level=Full Pool=ThursdayPool Thursday at 8:00pm
  Run = Level=Full Pool=WeeklyPool Friday at 8:00pm
}
# This does the catalog. It starts after the WeeklyCycle
Schedule {
  Name = "WeeklyCycleAfterBackup"
  Run = Level=Full Pool=MondayPool Monday at 8:15pm
  Run = Level=Full Pool=TuesdayPool Tuesday at 8:15pm
  Run = Level=Full Pool=WednesdayPool Wednesday at 8:15pm
  Run = Level=Full Pool=ThursdayPool Thursday at 8:15pm
  Run = Level=Full Pool=WeeklyPool Friday at 8:15pm
}
# This is the backup of the catalog
FileSet {
  Name = "Catalog"
  Include {
```

```
    Options {
      signature=MD5
    }
    File = /opt/bacula/working/bacula.sql
  }
}
# Client (File Services) to backup
Client {
  Name = ServerName
  Address = dionysus
  FDPort = 9102
  Catalog = MyCatalog
  Password = "client-pass"
  File Retention = 30d
  Job Retention = 30d
  AutoPrune = yes
}
# Definition of file storage device
Storage {
  Name = Tape
  Address = dionysus
  SDPort = 9103
  Password = "storage-pass"
  Device = Tandberg
  Media Type = MLR1
}
# Generic catalog service
Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}
# Reasonable message delivery -- send almost all to email address
#  and to the console
Messages {
  Name = Standard
  mailcommand = "/opt/bacula/bin/bsmtp -h localhost -f \"\(Bacula\) %r\"
     -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/opt/bacula/bin/bsmtp -h localhost -f \"\(Bacula\) %r\"
     -s \"Bacula: Intervention needed for %j\" %r"
  mail = root@localhost = all, !skipped
  operator = root@localhost = mount
  console = all, !skipped, !saved
  append = "/opt/bacula/log/bacula.log" = all, !skipped
}

# Pool definitions
#
# Default Pool for jobs, but will hold no actual volumes
Pool {
  Name = Default
  Pool Type = Backup
}
Pool {
  Name = MondayPool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 6d
  Maximum Volume Jobs = 2
}
Pool {
  Name = TuesdayPool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 6d
  Maximum Volume Jobs = 2
}
Pool {
  Name = WednesdayPool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 6d
  Maximum Volume Jobs = 2
```

```
}
Pool {
  Name = ThursdayPool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 6d
  Maximum Volume Jobs = 2
}
Pool {
  Name = WeeklyPool
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 12d
  Maximum Volume Jobs = 2
}
# EOF
```

Note, the mailcommand and operatorcommand should be on a single line each. They were split to preserve the proper page width. In order to get Bacula to release the tape after the nightly backup, he uses a **RunAfterJob** script that deletes the ASCII copy of the database back and then rewinds and ejects the tape. The following is a copy of `end\_of\_backup.sh`

```
#! /bin/sh
/opt/bacula/delete_catalog_backup
mt rewind
mt eject
exit 0
```

Finally, if you list his Volumes, you get something like the following:

```
*list media
Using default Catalog name=MyCatalog DB=bacula
Pool: WeeklyPool
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| MeId| VolumeName| MedTyp| VolStat| VolBytes  | LastWritten     | VolRet| Recyc|
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| 5   | Friday_1  | MLR1  | Used   | 2157171998| 2003-07-11 20:20| 103680| 1    |
| 6   | Friday_2  | MLR1  | Append | 0         | 0               | 103680| 1    |
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
Pool: MondayPool
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| MeId| VolumeName| MedTyp| VolStat| VolBytes  | LastWritten     | VolRet| Recyc|
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| 2   | Monday    | MLR1  | Used   | 2260942092| 2003-07-14 20:20| 518400| 1    |
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
Pool: TuesdayPool
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| MeId| VolumeName| MedTyp| VolStat| VolBytes  | LastWritten     | VolRet| Recyc|
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| 3   | Tuesday   | MLR1  | Used   | 2268180300| 2003-07-15 20:20| 518400| 1    |
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
Pool: WednesdayPool
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| MeId| VolumeName| MedTyp| VolStat| VolBytes  | LastWritten     | VolRet| Recyc|
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| 4   | Wednesday | MLR1  | Used   | 2138871127| 2003-07-09 20:2 | 518400| 1    |
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
Pool: ThursdayPool
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| MeId| VolumeName| MedTyp| VolStat| VolBytes  | LastWritten     | VolRet| Recyc|
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
| 1   | Thursday  | MLR1  | Used   | 2146276461| 2003-07-10 20:50| 518400| 1    |
+-----+-----------+-------+--------+-----------+-----------------+-------+------+
Pool: Default
No results to list.
```

Note, I have truncated a number of the columns so that the information fits on the width of a page.

Chapter 36

Autochanger Support

Bacula provides autochanger support for reading and writing tapes. In order to work with an autochanger, Bacula requires a number of things, each of which is explained in more detail after this list:

- A script that actually controls the autochanger according to commands sent by Bacula. We furnish such a script that works with `mtx` found in the **depkgs** distribution.
- That each Volume (tape) to be used must be defined in the Catalog and have a Slot number assigned to it so that Bacula knows where the Volume is in the autochanger. This is generally done with the `label` command, but can also done after the tape is labeled using the `update slots` command. See below for more details. You must pre-label the tapes manually before using them.
- Modifications to your Storage daemon's Device configuration resource to identify that the device is a changer, as well as a few other parameters.
- You should also modify your Storage resource definition in the Director's configuration file so that you are automatically prompted for the Slot when labeling a Volume.
- You need to ensure that your Storage daemon (if not running as root) has access permissions to both the tape drive and the control device.
- You need to have **Autochanger = yes** in your Storage resource in your `bacula-dir.conf` file so that you will be prompted for the slot number when you label Volumes.

In version 1.37 and later, there is a new Autochanger resource that permits you to group Device resources thus creating a multi-drive autochanger. If you have an autochanger, you **must** use this new resource.

Bacula uses its own `mtx-changer` script to interface with a program that actually does the tape changing. Thus in principle, `mtx-changer` can be adapted to function with any autochanger program, or you can call any other script or program. The current version of `mtx-changer` works with the `mtx` program. However, FreeBSD users have provided a script in the `examples/autochangers` directory that allows Bacula to use the `chio` program.

Bacula also supports autochangers with barcode readers. This support includes two Console commands: `label barcodes` and `update slots`. For more details on these commands, see the "Barcode Support" section below.

Current Bacula autochanger support does not include cleaning, stackers, or silos. Stackers and silos are not supported because Bacula expects to be able to access the slots randomly. However, if you are very careful to setup Bacula to access the Volumes in the autochanger sequentially, you may be able to make Bacula work with stackers (gravity feed and such).

Support for multi-drive autochangers requires the Autochanger resource introduced in version 1.37. This resource is also recommended for single drive autochangers.

In principle, if mtx will operate your changer correctly, then it is just a question of adapting the mtx-changer script (or selecting one already adapted) for proper interfacing.

Note, we have feedback from some users that there are certain incompatibilities between the Linux kernel and mtx. For example between kernel 2.6.18-8.1.8.el5 of CentOS and RedHat and version 1.3.10 and 1.3.11 of mtx. This was fixed by upgrading to a version 2.6.22 kernel.

In addition, apparently certain versions of mtx, for example, version 1.3.11 limit the number of slots to a maximum of 64. The solution was to use version 1.3.10.

If you are having troubles, please use the auto command in the btape program to test the functioning of your autochanger with Bacula. When Bacula is running, please remember that for many distributions (e.g. FreeBSD, Debian, ...) the Storage daemon runs as **bacula.tape** rather than **root.root**, so you will need to ensure that the Storage daemon has sufficient permissions to access the autochanger.

Some users have reported that the the Storage daemon blocks under certain circumstances in trying to mount a volume on a drive that has a different volume loaded. As best we can determine, this is simply a matter of waiting a bit. The drive was previously in use writing a Volume, and sometimes the drive will remain BLOCKED for a good deal of time (up to 7 minutes on a slow drive) waiting for the cassette to rewind and to unload before the drive can be used with a different Volume.

36.1 Knowing What SCSI Devices You Have

Under Linux, you can

```
cat /proc/scsi/scsi
```

to see what SCSI devices you have available. You can also:

```
cat /proc/scsi/sg/device_hdr /proc/scsi/sg/devices
```

to find out how to specify their control address (/dev/sg0 for the first, /dev/sg1 for the second, ...) on the **Changer Device =** Bacula directive.

You can also use the excellent lsscsi tool.

```
$ lsscsi -g
 [1:0:2:0]    tape    SEAGATE  ULTRIUM06242-XXX 1619  /dev/st0  /dev/sg9
 [1:0:14:0]   mediumx STK      L180             0315  /dev/sch0 /dev/sg10
 [2:0:3:0]    tape    HP       Ultrium 3-SCSI   G24S  /dev/st1  /dev/sg11
 [3:0:0:0]    enclosu HP       A6255A           HP04  -         /dev/sg3
 [3:0:1:0]    disk    HP 36.4G ST336753FC       HP00  /dev/sdd  /dev/sg4
```

For more detailed information on what SCSI devices you have please see the **Linux SCSI Tricks** section (section 3.2.2 page 30) of the **Tape Testing** chapter (chapter 3 page 25) of the Bacula Community Edition Problems Resolution Guide.

Under FreeBSD, you can use:

```
camcontrol devlist
```

To list the SCSI devices as well as the /dev/passn that you will use on the Bacula **Changer Device =** directive.

Please check that your Storage daemon has permission to access this device.

The following tip for FreeBSD users comes from Danny Butroyd: on reboot Bacula will NOT have permission to control the device /dev/pass0 (assuming this is your changer device). To get around this just edit the /etc/devfs.conf file and add the following to the bottom:

```
own     pass0   root:bacula
perm    pass0   0666
own     nsa0.0  root:bacula
perm    nsa0.0  0666
```

This gives the bacula group permission to write to the nsa0.0 device too just to be on the safe side. To bring these changes into effect just run:

```
/etc/rc.d/devfs restart
```

Basically this will stop you having to manually change permissions on these devices to make Bacula work when operating the AutoChanger after a reboot.

36.2 Example Scripts

Please read the sections below so that you understand how autochangers work with Bacula. Although we supply a default mtx-changer script, your autochanger may require some additional changes. If you want to see examples of configuration files and scripts, please look in the <bacula-src>/examples/devices directory where you will find an example HP-autoloader.conf Bacula Device resource, and several mtx-changer scripts that have been modified to work with different autochangers.

36.3 Slots

To properly address autochangers, Bacula must know which Volume is in each **slot** of the autochanger. Slots are where the changer cartridges reside when not loaded into the drive. Bacula numbers these slots from one to the number of cartridges contained in the autochanger.

Bacula will not automatically use a Volume in your autochanger unless it is labeled and the slot number is stored in the catalog and the Volume is marked as InChanger. This is because it must know where each volume is (slot) to be able to load the volume.

For each Volume in your changer, you will, using the Console program, assign a slot. This information is kept in **Bacula's** catalog database along with the other data for the volume. If no slot is given, or the slot is set to zero, Bacula will not attempt to use the autochanger even if all the necessary configuration records are present. When doing a mount command on an autochanger, you must specify which slot you want mounted. If the drive is loaded with a tape from another slot, it will unload it and load the correct tape, but normally, no tape will be loaded because an unmount command causes Bacula to unload the tape in the drive.

You can check if the Slot number and InChanger flag are set by doing a:

```
list Volumes
```

in the Console program.

36.4 Multiple Devices

Some autochangers have more than one read/write device (drive). The new Autochanger resource introduced in version 1.37 permits you to group Device resources, where each device represents a drive. The Director may still reference the Devices (drives) directly, but doing so, bypasses the proper functioning of the drives together. Instead, the Director (in the Storage resource) should reference the Autochanger resource name. Doing so permits the Storage daemon to ensure that only one drive uses the `mtx-changer` script at a time, and also that two drives don't reference the same Volume.

Multi-drive requires the use of the **Drive Index** directive in the `Device` resource of the Storage daemon's configuration file. Drive numbers or the Device Index are numbered beginning at zero (0), which is the default. To use the second Drive in an autochanger, you need to define a second `Device` resource and set the **Drive Index** to 1 for that device. In general, the second device will have the same **Changer Device** (control channel) as the first drive, but a different **Archive Device**.

As a default, Bacula jobs will prefer to write to a Volume that is already mounted. If you have a multiple drive autochanger and you want Bacula to write to more than one Volume in the same Pool at the same time, you will need to set Prefer Mounted Volumes in the Directors Job resource to **no**. This will cause the Storage daemon to maximize the use of drives.

36.5 Device Configuration Records

Configuration of autochangers within Bacula is done in the Device resource of the Storage daemon. Four records: **Autochanger**, **Changer Device**, **Changer Command**, and **Maximum Changer Wait** control how Bacula uses the autochanger.

These four records, permitted in **`Device`** resources, are described in detail below. Note, however, that the **Changer Device** and the **Changer Command** directives are not needed in the `Device` resource if they are present in the **`Autochanger`** resource.

Autochanger = <yes|no> The **Autochanger** record specifies that the current device is or is not an autochanger. The default is no.

Changer Device = <device-name> In addition to the Archive Device name, you must specify a **Changer Device** name. This is because most autochangers are controlled through a different device than is used for reading and writing the cartridges. For example, on Linux, one normally uses the generic SCSI interface for controlling the autochanger, but the standard SCSI interface for reading and writing the tapes. On Linux, for the **Archive Device = `/dev/nst0`**, you would typically have **Changer Device = `/dev/sg0`**. Note, some of the more advanced autochangers will locate the changer device on `/dev/sg1`. Such devices typically have several drives and a large number of tapes.

On FreeBSD systems, the changer device will typically be on `/dev/pass0` through `/dev/passn`.

On Solaris, the changer device will typically be some file under **`/dev/rdsk`**.

Please ensure that your Storage daemon has permission to access this device.

Changer Command = <command> This record is used to specify the external program to call and what arguments to pass to it. The command is assumed to be a standard program or shell script that can be executed by the operating system. This command is invoked each time that Bacula wishes to manipulate the autochanger. The following substitutions are made in the **command** before it is sent to the operating system for execution:

```
%% = %
%a = archive device name
%c = changer device name
```

```
%d = changer drive index base 0
%f = Client's name
%j = Job name
%o = command  (loaded, load, or unload)
%s = Slot base 0
%S = Slot base 1
%v = Volume name
```

An actual example for using `mtx` with the `mtx-changer` script (part of the Bacula distribution) is:

```
Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
```

Where you will need to adapt the `/opt/bacula` to be the actual path on your system where the `mtx-changer` script resides. Details of the three commands currently used by Bacula (`loaded`, `load`, `unload`) as well as the output expected by Bacula are give in the **Bacula Autochanger Interface** section below.

Maximum Changer Wait = <time> This record is used to define the maximum amount of time that Bacula will wait for an autoloader to respond to a command (e.g. `load`). The default is set to 120 seconds. If you have a slow autoloader you may want to set it longer.

If the autoloader program fails to respond in this time, it will be killed and Bacula will request operator intervention.

Drive Index = <number> This record allows you to tell Bacula to use the second or subsequent drive in an autochanger with multiple drives. Since the drives are numbered from zero, the second drive is defined by

```
Drive Index = 1
```

To use the second drive, you need a second Device resource definition in the Bacula configuration file. See the Multiple Drive section above in this chapter for more information.

In addition, for proper functioning of the Autochanger, you must define an `Autochanger` resource.

Chapter 37

Autochanger Resource

The Autochanger resource supports single or multiple drive autochangers by grouping one or more Device resources into one unit called an autochanger in Bacula (often referred to as a "tape library" by autochanger manufacturers).

If you have an Autochanger, and you want it to function correctly, you **must** have an Autochanger resource in your Storage conf file, and your Director's Storage directives that want to use an Autochanger **must** refer to the Autochanger resource name. In previous versions of Bacula, the Director's Storage directives referred directly to Device resources that were autochangers. In version 1.38.0 and later, referring directly to Device resources will not work for Autochangers.

Name = <Autochanger-Name> Specifies the Name of the Autochanger. This name is used in the Director's Storage definition to refer to the autochanger. This directive is required.

Device = <Device-name1, device-name2, . . . > Specifies the names of the Device resource or resources that correspond to the autochanger drive. If you have a multiple drive autochanger, you must specify multiple Device names, each one referring to a separate Device resource that contains a Drive Index specification that corresponds to the drive number base zero. You may specify multiple device names on a single line separated by commas, and/or you may specify multiple Device directives. This directive is required.

Changer Device = <name-string> The specified <**name-string**> gives the system file name of the autochanger device name. If specified in this resource, the Changer Device name is not needed in the Device resource. If it is specified in the Device resource (see above), it will take precedence over one specified in the Autochanger resource.

Changer Command = <name-string> The <**name-string**> specifies an external program to be called that will automatically change volumes as required by **Bacula**. Most frequently, you will specify the Bacula supplied `mtx-changer` script as follows. If it is specified here, it need not be specified in the Device resource. If it is also specified in the Device resource, it will take precedence over the one specified in the Autochanger resource.

The following is an example of a valid Autochanger resource definition:

```
Autochanger {
  Name = "DDS-4-changer"
  Device = DDS-4-1, DDS-4-2, DDS-4-3
  Changer Device = /dev/sg0
  Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}
Device {
  Name = "DDS-4-1"
  Drive Index = 0
  Autochanger = yes
  ...
```

```
}
Device {
  Name = "DDS-4-2"
  Drive Index = 1
  Autochanger = yes
  ...
Device {
  Name = "DDS-4-3"
  Drive Index = 2
  Autochanger = yes
  Autoselect = no
  ...
}
```

Please note that it is important to include the **Autochanger = yes** directive in each Device definition that belongs to an Autochanger. A device definition should not belong to more than one Autochanger resource. Also, your Device directive in the Storage resource of the Director's conf file should have the Autochanger's resource name rather than a name of one of the Devices.

If you have a drive that physically belongs to an Autochanger but you don't want to have it automatically used when Bacula references the Autochanger for backups, for example, you want to reserve it for restores, you can add the directive:

```
Autoselect = no
```

to the Device resource for that drive. In that case, Bacula will not automatically select that drive when accessing the Autochanger. You can, still use the drive by referencing it by the Device name directly rather than the Autochanger name. An example of such a definition is shown above for the Device DDS-4-3, which will not be selected when the name DDS-4-changer is used in a Storage definition, but will be used if DDS-4-3 is used.

37.1 An Example Configuration File

The following two resources implement an autochanger:

```
Autochanger {
  Name = "Autochanger"
  Device = DDS-4
  Changer Device = /dev/sg0
  Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}

Device {
  Name = DDS-4
  Media Type = DDS-4
  Archive Device = /dev/nst0    # Normal archive device
  Autochanger = yes
  LabelMedia = no
  AutomaticMount = yes
  AlwaysOpen = yes
}
```

where you will adapt the **Archive Device**, the **Changer Device**, and the path to the **Changer Command** to correspond to the values used on your system.

37.2 A Multi-drive Example Configuration File

The following resources implement a multi-drive autochanger:

```
Autochanger {
  Name = "Autochanger"
  Device = Drive-1,Drive-2
  Changer Device = /dev/sg0
  Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}

Device {
  Name = Drive-1
  Drive Index = 0
  Media Type = DDS-4
  Archive Device = /dev/nst0    # Normal archive device
  Autochanger = yes
  LabelMedia = no
  AutomaticMount = yes
  AlwaysOpen = yes
}

Device {
  Name = Drive-2
  Drive Index = 1
  Media Type = DDS-4
  Archive Device = /dev/nst1    # Normal archive device
  Autochanger = yes
  LabelMedia = no
  AutomaticMount = yes
  AlwaysOpen = yes
}
```

where you will adapt the **Archive Device**, the **Changer Device**, and the path to the **Changer Command** to correspond to the values used on your system.

37.3 Specifying Slots When Labeling

If you add an **Autochanger = yes** record to the Storage resource in your Director's configuration file, the Bacula Console will automatically prompt you for the slot number when the Volume is in the changer when you `add` or `label` tapes for that Storage device. If your `mtx-changer` script is properly installed, Bacula will automatically load the correct tape during the `label` command.

You must also set **Autochanger = yes** in the Storage daemon's Device resource as we have described above in order for the autochanger to be used. Please see the Storage Resource in the Director's chapter and the Device Resource in the Storage daemon chapter for more details on these records.

Thus all stages of dealing with tapes can be totally automated. It is also possible to set or change the Slot using the `update` command in the Console and selecting **Volume Parameters** to update.

Even though all the above configuration statements are specified and correct, Bacula will attempt to access the autochanger only if a **slot** is non-zero in the catalog Volume record (with the Volume name).

If your autochanger has barcode labels, you can label all the Volumes in your autochanger one after another by using the `label barcodes` command. For each tape in the changer containing a barcode, Bacula will mount the tape and then label it with the same name as the barcode. An appropriate Media record will also be created in the catalog. Any barcode that begins with the same characters as specified on the "**CleaningPrefix=xxx** command, will be treated as a cleaning tape, and will not be labeled.

For example with

```
Pool {
  Name ...
```

```
  Cleaning Prefix = "CLN"
}
```

any slot containing a barcode of CLNxxxx will be treated as a cleaning tape and will not be mounted.

Please note that Volumes must be pre-labeled to be automatically used in the autochanger during a backup. If you do not have a barcode reader, this is done manually (or via a script).

37.4 Changing Cartridges

If you wish to insert or remove cartridges in your autochanger or you manually run the mtx program, you must first tell Bacula to release the autochanger by doing:

```
unmount
```

Change cartridges and/or run mtx

```
mount
```

If you do not do the unmount before making such a change, Bacula will become completely confused about what is in the autochanger and may stop function because it expects to have exclusive use of the autochanger while it has the drive mounted.

37.5 Dealing with Multiple Magazines

If you have several magazines or if you insert or remove cartridges from a magazine, you should notify Bacula of this. By doing so, Bacula will as a preference, use Volumes that it knows to be in the autochanger before accessing Volumes that are not in the autochanger. This prevents unneeded operator intervention.

If your autochanger has barcodes (machine readable tape labels), the task of informing Bacula is simple. Every time, you change a magazine, or add or remove a cartridge from the magazine, simply do

```
unmount
```

Remove magazine

Insert new magazine

```
update slots
mount
```

in the Console program. This will cause Bacula to request the autochanger to return the current Volume names in the magazine. This will be done without actually accessing or reading the Volumes because the barcode reader does this during inventory when the autochanger is first turned on. Bacula will ensure that any Volumes that are currently marked as being in the magazine are marked as no longer in the magazine, and the new list of Volumes will be marked as being in the magazine. In addition, the Slot numbers of the Volumes will be corrected in Bacula's catalog if they are incorrect (added or moved).

If you do not have a barcode reader on your autochanger, you have several alternatives.

1. You can manually set the Slot and InChanger flag using the `update volume` command in the Console (quite painful).
2. You can issue a

   ```
   update slots scan
   ```

   command that will cause Bacula to read the label on each of the cartridges in the magazine in turn and update the information (Slot, InChanger flag) in the catalog. This is quite effective but does take time to load each cartridge into the drive in turn and read the Volume label.
3. You can modify the `mtx-changer` script so that it simulates an autochanger with barcodes. See below for more details.

37.6 Simulating Barcodes in your Autochanger

You can simulate barcodes in your autochanger by making the `mtx-changer` script return the same information that an autochanger with barcodes would do. This is done by commenting out the one and only line in the **list)** case, which is:

```
${MTX} -f $ctl status | grep " *Storage Element [0-9]*:.*Full" | awk "{print \$3 \$4}"
                      | sed "s/Full *\(:VolumeTag=\)*//"
```

at approximately line 99 by putting a # in column one of that line, or by simply deleting it. Then in its place add a new line that prints the contents of a file. For example:

```
cat /etc/bacula/changer.volumes
```

Be sure to include a full path to the file, which can have any name. The contents of the file must be of the following format:

```
1:Volume1
2:Volume2
3:Volume3
...
```

Where the 1, 2, 3 are the slot numbers and Volume1, Volume2, ... are the Volume names in those slots. You can have multiple files that represent the Volumes in different magazines, and when you change magazines, simply copy the contents of the correct file into your `/etc/bacula/changer.volumes` file. There is no need to stop and start Bacula when you change magazines, simply put the correct data in the file, then run the `update slots` command, and your autochanger will appear to Bacula to be an autochanger with barcodes.

37.7 The Full Form of the Update Slots Command

If you change only one cartridge in the magazine, you may not want to scan all Volumes, so the `update slots` command (as well as the `update slots scan` command) has the additional form:

```
update slots=n1,n2,n3-n4, ...
```

where the keyword **scan** can be appended or not. The n1,n2, ... represent Slot numbers to be updated and the form n3-n4 represents a range of Slot numbers to be updated (e.g. 4-7 will update Slots 4,5,6, and 7).

This form is particularly useful if you want to do a scan (time expensive) and restrict the update to one or two slots.

For example, the command:

```
update slots=1,6 scan
```

will cause Bacula to load the Volume in Slot 1, read its Volume label and update the Catalog. It will do the same for the Volume in Slot 6. The command:

```
update slots=1-3,6
```

will read the barcoded Volume names for slots 1,2,3 and 6 and make the appropriate updates in the Catalog. If you don't have a barcode reader or have not modified the mtx-changer script as described above, the above command will not find any Volume names so will do nothing.

37.8 FreeBSD Issues

If you are having problems on FreeBSD when Bacula tries to select a tape, and the message is **Device not configured**, this is because FreeBSD has made the tape device /dev/nsa1 disappear when there is no tape mounted in the autochanger slot. As a consequence, Bacula is unable to open the device. The solution to the problem is to make sure that some tape is loaded into the tape drive before starting Bacula. This problem is corrected in Bacula versions 1.32f-5 and later.

Please see the **Tape Testing** chapter (chapter 3.3.6 page 36) of the Bacula Community Edition Problems Resolution Guide for **important** information concerning your tape drive before doing the autochanger testing.

37.9 Testing Autochanger and Adapting mtx-changer script

Before attempting to use the autochanger with Bacula, it is preferable to "hand-test" that the changer works. To do so, we suggest you do the following commands (assuming that the mtx-changer script is installed in /opt/bacula/scripts/mtx-changer):

Make sure Bacula is not running.

/opt/bacula/scripts/mtx-changer /dev/sg0 list 0 /dev/nst0 0 This command should print:

```
1:
2:
3:
...
```

or one number per line for each slot that is occupied in your changer, and the number should be terminated by a colon (:). If your changer has barcodes, the barcode will follow the colon. If an error message is printed, you must resolve the problem (e.g. try a different SCSI control device name if /dev/sg0 is incorrect). For example, on FreeBSD systems, the autochanger SCSI control device is generally /dev/pass2.

/opt/bacula/scripts/mtx-changer /dev/sg0 listall 0 /dev/nst0 0 This command should print:

```
Drive content:          D:Drive num:F:Slot loaded:Volume Name
D:0:F:2:vol2        or D:Drive num:E
D:1:F:42:vol42
D:3:E

Slot content:
S:1:F:vol1              S:Slot num:F:Volume Name
S:2:E               or S:Slot num:E
S:3:F:vol4

Import/Export tray slots:
I:10:F:vol10            I:Slot num:F:Volume Name
I:11:E              or I:Slot num:E
I:12:F:vol40
```

/opt/bacula/scripts/mtx-changer /dev/sg0 transfer 1 2 This command should transfer a volume from source (1) to destination (2)

/opt/bacula/scripts/mtx-changer /dev/sg0 slots This command should return the number of slots in your autochanger.

/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0 If a tape is loaded from slot 1, this should cause it to be unloaded.

/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0 Assuming you have a tape in slot 3, it will be loaded into drive (0).

/opt/bacula/scripts/mtx-changer /dev/sg0 loaded 0 /dev/nst0 0 It should print "3" Note, we have used an "illegal" slot number 0. In this case, it is simply ignored because the slot number is not used. However, it must be specified because the drive parameter at the end of the command is needed to select the correct drive.

/opt/bacula/scripts/mtx-changer /dev/sg0 unload 3 /dev/nst0 0 will unload the tape into slot 3.

Once all the above commands work correctly, assuming that you have the right **Changer Command** in your configuration, Bacula should be able to operate the changer. The only remaining area of problems will be if your autoloader needs some time to get the tape loaded after issuing the command. After the `mtx-changer` script returns, Bacula will immediately rewind and read the tape. If Bacula gets rewind I/O errors after a tape change, you will probably need to insert a **sleep 20** after the `mtx` command, but be careful to exit the script with a zero status by adding **exit 0** after any additional commands you add to the script. This is because Bacula checks the return status of the script, which should be zero if all went well.

You can test whether or not you need a **sleep** by putting the following commands into a file and running it as a script:

```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

If the above script runs, you probably have no timing problems. If it does not run, start by putting a **sleep 30** or possibly a **sleep 60** in the script just after the `mtx-changer` load command. If that works, then you should move the sleep into the actual `mtx-changer` script so that it will be effective when Bacula runs.

A second problem that comes up with a small number of autochangers is that they need to have the cartridge ejected before it can be removed. If this is the case, the **load 3** will never succeed regardless of how long you wait. If this seems to be your problem, you can insert an eject just after the unload so that the script looks like:

```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
mt -f /dev/st0 offline
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

Obviously, if you need the offline command, you should move it into the mtx-changer script ensuring that you save the status of the mtx command or always force an **exit 0** from the script, because Bacula checks the return status of the script.

As noted earlier, there are several scripts in <bacula-source>/examples/devices that implement the above features, so they may be a help to you in getting your script to work.

If Bacula complains "Rewind error on /dev/nst0. ERR=Input/output error." you most likely need more sleep time in your mtx-changer before returning to Bacula after a load command has been completed.

37.10 Using the Autochanger

Let's assume that you have properly defined the necessary Storage daemon Device records, and you have added the **Autochanger = yes** record to the Storage resource in your Director's configuration file.

Now you fill your autochanger with say six blank tapes.

What do you do to make Bacula access those tapes?

One strategy is to prelabel each of the tapes. Do so by starting Bacula, then with the Console program, enter the label command:

```
./bconsole
Connecting to Director rufus:9101
1000 OK: rufus-dir Version: 1.26 (4 October 2002)
*label
```

it will then print something like:

```
Using default Catalog name=BackupDB DB=bacula
The defined Storage resources are:
     1: Autochanger
     2: File
Select Storage resource (1-2): 1
```

I select the autochanger (1), and it prints:

```
Enter new Volume name: TestVolume1
Enter slot (0 for none): 1
```

where I entered **TestVolume1** for the tape name, and slot **1** for the slot. It then asks:

```
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2): 1
```

I select the Default pool. This will be automatically done if you only have a single pool, then Bacula will proceed to unload any loaded volume, load the volume in slot 1 and label it. In this example, nothing was in the drive, so it printed:

```
Connecting to Storage daemon Autochanger at localhost:9103 ...
Sending label command ...
3903 Issuing autochanger "load slot 1" command.
3000 OK label. Volume=TestVolume1 Device=/dev/nst0
Media record for Volume=TestVolume1 successfully created.
Requesting mount Autochanger ...
3001 Device /dev/nst0 is mounted with Volume TestVolume1
You have messages.
*
```

You may then proceed to label the other volumes. The messages will change slightly because Bacula will unload the volume (just labeled TestVolume1) before loading the next volume to be labeled.

Once all your Volumes are labeled, Bacula will automatically load them as they are needed.

To "see" how you have labeled your Volumes, simply enter the `list volumes` command from the Console program, which should print something like the following:

```
  * list volumes
Using default Catalog name=BackupDB DB=bacula
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2): 1
+-------+----------+---------+---------+-------+--------+----------+-------+------+
| MedId | VolName  | MedTyp | VolStat | Bites | LstWrt | VolReten | Recyc | Slot |
+-------+----------+---------+---------+-------+--------+----------+-------+------+
| 1     | TestVol1 | DDS-4   | Append  | 0     | 0      | 30672000 | 0     | 1    |
| 2     | TestVol2 | DDS-4   | Append  | 0     | 0      | 30672000 | 0     | 2    |
| 3     | TestVol3 | DDS-4   | Append  | 0     | 0      | 30672000 | 0     | 3    |
| ...                                                                          |
+-------+----------+---------+---------+-------+--------+----------+-------+------+
```

37.11 Barcode Support

Bacula provides barcode support with two Console commands, `label barcodes` and `update slots`.

The `label barcodes` will cause Bacula to read the barcodes of all the cassettes that are currently installed in the magazine (cassette holder) using the `mtx-changer` **list** command. Each cassette is mounted in turn and labeled with the same Volume name as the barcode.

The `update slots` command will first obtain the list of cassettes and their barcodes from `mtx-changer`. Then it will find each volume in turn in the catalog database corresponding to the barcodes and set its slot to correspond to the value just read. If the Volume is not in the catalog, then nothing will be done. This command is useful for synchronizing Bacula with the current magazine in case you have changed magazines or in case you have moved cassettes from one slot to another. If the autochanger is empty, nothing will be done.

The **Cleaning Prefix** statement can be used in the Pool resource to define a Volume name prefix, which if it matches that of the Volume (barcode) will cause that Volume to be marked with a VolStatus of **Cleaning**. This will prevent Bacula from attempting to write on the Volume.

37.12 Use bconsole to display Autochanger content

The `status slots storage=xxx` command displays autochanger content.

```
 Slot |  Volume Name    |  Status  |      Type          |    Pool         | Loaded |
------+-----------------+----------+--------------------+-----------------+--------|
    1 |           00001 |   Append |  DiskChangerMedia  |         Default |    0   |
    2 |           00002 |   Append |  DiskChangerMedia  |         Default |    0   |
    3*|           00003 |   Append |  DiskChangerMedia  |         Scratch |    0   |
    4 |                 |          |                    |                 |    0   |
```

If you see a * near the slot number, you have to run `update slots` command to synchronize autochanger content with your catalog.

37.13 Bacula Autochanger Interface

Bacula calls the autochanger script that you specify on the **Changer Command** statement. Normally this script will be the `mtx-changer` script that we provide, but it can in fact be any program. The only requirement for the script is that it must understand the commands that Bacula uses, which are **loaded**, **load**, **unload**, **list**, and **slots**. In addition, each of those commands must return the information in the precise format as specified below:

Currently the changer commands used are:

loaded returns number of the slot that is loaded, base 1, in the drive or 0 if the drive is empty.

load loads a specified slot (note, some autochangers require a 30 second pause after this command) into the drive.

unload unloads the device (returns cassette to its slot).

list returns one line for each cassette in the autochanger in the format <slot>:<barcode>. Where the <slot> is the non-zero integer representing the slot number, and <barcode> is the barcode associated with the cassette if it exists and if you autoloader supports barcodes. Otherwise the barcode field is blank.

slots returns total number of slots in the autochanger.

Bacula checks the exit status of the program called, and if it is zero, the data is accepted. If the exit status is non-zero, Bacula will print an error message and request the tape be manually mounted on the drive.

Chapter 38

Supported Autochangers

I hesitate to call these "supported" autochangers because the only autochangers that I have in my possession and am able to test are the HP SureStore DAT40X6 and the Overland PowerLoader LTO-2. All the other autochangers have been reported to work by Bacula users. Note, in the Capacity/Slot column below, I quote the Compressed capacity per tape (or Slot).

Since on most systems (other than FreeBSD), Bacula uses `mtx` through the `mtx-changer` script, in principle, if `mtx` will operate your changer correctly, then it is just a question of adapting the `mtx-changer` script (or selecting one already adapted) for proper interfacing.

Table 38.1: Autochangers known to work with Bacula

O.S.	Manuf.	Media	Model	Slots	Cap / Slot
Linux	Adic	DDS-3	Adic 1200G	12	—
Linux	Adic	DLT	FastStore 4000	7	20GB
Linux	Adic	LTO-1/2, SDLT 320	Adic Scalar 24	24	100GB
Linux	Adic	LTO-2	Adic FastStor 2, Sun Storedge L8	8	200GB
Linux	BDT	AIT	BDT ThinStor	?	200GB
—	CA-VM	Unknown	Tape	Unknown	Unknown
Linux	Dell	DLT VI,LTO-2,LTO3	PowerVault 122T/132T/136T	—	100GB
Linux	Dell	LTO-2	PowerVault 124T	—	200GB
—	DFSMS	Unknown	VM RMM	—	Unknown
Linux	Exabyte	VXA2	VXA PacketLoader 1x10 2U	10	80/160GB
—	Exabyte	LTO	Magnum 1x7 LTO Tape Auotloader	7	200/400GB
Linux	Exabyte	AIT-2	215A	15 (2 drives)	50GB
Linux	HP	DDS-4	SureStore DAT-40X6	6	40GB
Linux	HP	Ultrium-2/LTO	MSL 6000/ 60030/ 5052	28	200/400GB
—	HP	DLT	A4853 DLT	30	40/70GB

Continues on the following page

[]*Cont.*

O.S.	Manuf.	Media	Model	Slots	Cap / Slot
Linux	HP (Compaq)	DLT VI	Compaq TL-895	96+4 import export	35/70GB
z/VM	IBM	Unknown	IBM Tape Manager	—	Unknown
z/VM	IBM	Unknown	native tape	—	Unknown
Linux	IBM	LTO	IBM 3581 Ultrium Tape Loader	7	200/400GB
FreeBSD 5.4	IBM	DLT	IBM 3502-R14 – rebranded ATL L-500	14	35/70GB
Linux	IBM	Unknown?	IBM TotalStorage 3582L23	Unknown	Unknown
Debian	Overland	LTO	Overland LoaderXpress LTO/DLT8000	10-19	40-100GB
Fedora	Overland	LTO	Overland PowerLoader LTO-2	10-19	200/400GB
FreeBSD 5.4-Stable	Overland	LTO-2	Overland Powerloader tape	17	100GB
—	Overland	LTO	Overland Neo2000 LTO	26-30	100GB
Linux	Quantum	DLT-S4	Superloader 3	16	800/1600GB
Linux	Quantum	LTO-2	Superloader 3	16	200/400GB
Linux	Quantum	LTO-3	PX502	Unknown	Unknown
FreeBSD 4.9	QUALSTAR TLS-4210 (Qualstar)	AIT1: 36GB, AIT2: 50GB all uncomp	QUALSTAR TLS-4210	12	AIT1: 36GB, AIT2: 50GB all uncomp
Linux	Skydata	DLT	ATL-L200	8	40/80
-	Sony	DDS-4	TSL-11000	8	40GB

Continues on the following page

[]Cont.

O.S.	Manuf.	Media	Model	Slots	Cap / Slot
Linux	Sony	AIT-2	LIB-304(SDX-500C)	?	200GB
Linux	Sony	AIT-3	LIB-D81)	?	200GB
FreeBSD 4.9-STABLE	Sony	AIT-1	TSL-SA300C	4	45/70GB
—	Storagetek	DLT	Timberwolf DLT	6	40/70
—	Storagetek	Unknown	ACSLS	Unknown	Unknown
Solaris	Sun	4mm DLT	Sun Desktop Archive Python 29279	4	20GB
Linux	Tandberg	DLT VI	VS 640	8[1]	35/70GB
Linux 2.6.x	Tandberg Data	SLR100	SLR100 Autoloader	8	50/100GB

[1] To be verified

Chapter 39

Data Spooling

Bacula allows you to specify that you want the Storage daemon to initially write your data to disk and then subsequently to tape. This serves several important purposes.

- It takes a long time for data to come in from the File daemon during an Incremental backup. If it is directly written to tape, the tape will start and stop or shoe-shine as it is often called causing tape wear. By first writing the data to disk, then writing it to tape, the tape can be kept in continual motion.
- While the spooled data is being written to the tape, the despooling process has exclusive use of the tape. This means that you can spool multiple simultaneous jobs to disk, then have them very efficiently despooled one at a time without having the data blocks from several jobs intermingled, thus substantially improving the time needed to restore files. While despooling, all jobs spooling continue running.
- Writing to a tape can be slow. By first spooling your data to disk, you can often reduce the time the File daemon is running on a system, thus reducing downtime, and/or interference with users. Of course, if your spool device is not large enough to hold all the data from your File daemon, you may actually slow down the overall backup.

Data spooling is exactly that "spooling". It is not a way to first write a "backup" to a disk file and then to a tape. When the backup has only been spooled to disk, it is not complete yet and cannot be restored until it is written to tape.

Bacula version 1.39.x and later supports writing a backup to disk then later **Migrating** or moving it to a tape (or any other medium). For details on this, please see the Migration chapter of this manual for more details.

The remainder of this chapter explains the various directives that you can use in the spooling process.

39.1 Data Spooling Directives

The following directives can be used to control data spooling.

- To turn data spooling on/off at the Job level in the Job resource in the Director's conf file (default no).
  SpoolData = <**yes**|**no**>
- To override the Job specification in a Schedule Run directive in the Director's conf file.
  SpoolData = <**yes**|**no**>

- To override the Job specification in a bconsole session using the run command. Please note that this does **not** refer to a configuration statement, but to an argument for the run command.
  SpoolData= <**yes**|**no**>
- To limit the the maximum spool file size for a particular job in the Job resource
  Spool Size = size Where size is a the maximum spool size for this job specified in bytes.
- To limit the maximum total size of the spooled data for a particular device. Specified in the Device resource of the Storage daemon's conf file (default unlimited).
  Maximum Spool Size = size Where size is a the maximum spool size for all jobs specified in bytes.
- To limit the maximum total size of the spooled data for a particular device for a single job. Specified in the Device Resource of the Storage daemon's configuration file (default unlimited).
  Maximum Job Spool Size = size Where size is the maximum spool file size for a single job specified in bytes.
- To specify the spool directory for a particular device. Specified in the Device Resource of the Storage daemon's configuration file (default, the working directory).
  Spool Directory = directory

39.2 FileSet consideration

Please be very careful to exclude the spool directory from any backup, otherwise, your job will write enormous amounts of data to the Volume, and most probably terminate in error. This is because in attempting to backup the spool file, the backup data will be written a second time to the spool file, and so on *ad infinitum*.

Another advice is to always specify the maximum spool size so that your disk doesn't completely fill up. In principle, data spooling will properly detect a full disk, and despool data allowing the job to continue. However, attribute spooling is not so kind to the user. If the disk on which attributes are being spooled fills, the job will be canceled. In addition, if your working directory is on the same partition as the spool directory, then Bacula jobs will fail possibly in bizarre ways when the spool fills.

39.3 Other Points

- When data spooling is enabled, Bacula automatically turns on attribute spooling. In other words, it also spools the catalog entries to disk. This is done so that in case the job fails, there will be no catalog entries pointing to non-existent tape backups.
- Attribute despooling occurs near the end of a job. The Storage daemon accumulates file attributes during the backup and sends them to the Director at the end of the job. The Director then inserts the file attributes into the catalog. During this insertion, the tape drive may be inactive. When the file attribute insertion is completed, the job terminates.
- Attribute spool files are always placed in the working directory of the Storage daemon.
- When Bacula begins despooling data spooled to disk, it takes exclusive use of the tape. This has the major advantage that in running multiple simultaneous jobs at the same time, the blocks of several jobs will not be intermingled.
- It probably does not make a lot of sense to enable data spooling if you are writing to disk files.

- It is probably best to provide as large a spool file as possible to avoid repeatedly spooling/despooling. Also, while a job is despooling to tape, the File daemon must wait (i.e. spooling stops for the job while it is despooling).
- If you are running multiple simultaneous jobs, Bacula will continue spooling other jobs while one is despooling to tape, provided there is sufficient spool file space.

Chapter 40

Using Bacula catalog to grab information

Bacula catalog contains lot of information about your IT infrastructure, how many files, their size, the number of video or music files etc. Using Bacula catalog during the day to get them permit to save resources on your servers.

In this chapter, you will find tips and information to measure bacula efficiency and report statistics.

40.1 Job statistics

If you (or probably your boss) want to have statistics on your backups to provide some Service Level Agreement (SLA) indicators, you could use a few SQL queries on the `Job` table to report how many:

- jobs have run
- jobs have been successful
- files have been backed up
- ...

However, these statistics are accurate only if your job retention is greater than your statistics period. I.e, if jobs are purged from the catalog, you won't be able to use them.

Now, you can use the `update stats [days=num]` console command to fill the `JobHisto` table with new Job records. If you want to be sure to take in account only **good jobs**, ie if one of your important job has failed but you have fixed the problem and restarted it on time, you probably want to delete the first *bad* job record and keep only the successful one. For that simply let your staff do the job, and update `JobHisto` table after two or three days depending on your organization using the **[days=num]** option.

These statistics records aren't used for restoring, but mainly for capacity planning, billings, etc.

The BWeb interface provides a statistics module that can use this feature. You can also use tools like Talend or extract information by yourself.

The **Statistics Retention** = <**time**> director directive defines the length of time that Bacula will keep statistics job records in the Catalog

database after the Job End time. (In JobHisto table) When this time period expires, and if user runs prune stats command, Bacula will prune (remove) Job records that are older than the specified period.

You can use the following Job resource in your nightly **BackupCatalog** job to maintain statistics.

```
Job {
  Name = BackupCatalog
  ...
  RunScript {
    Console = "update stats days=3"
    Console = "prune stats yes"
    RunsWhen = After
    RunsOnClient = no
  }
}
```

Chapter 41

ANSI and IBM Tape Labels

Bacula supports ANSI or IBM tape labels as long as you enable it. In fact, with the proper configuration, you can force Bacula to require ANSI or IBM labels.

Bacula can create an ANSI or IBM label, but if Check Labels is enabled (see below), Bacula will look for an existing label, and if it is found, it will keep the label. Consequently, you can label the tapes with programs other than Bacula, and Bacula will recognize and support them.

Even though Bacula will recognize and write ANSI and IBM labels, it always writes its own tape labels as well.

When using ANSI or IBM tape labeling, you must restrict your Volume names to a maximum of six characters.

If you have labeled your Volumes outside of Bacula, then the ANSI/IBM label will be recognized by Bacula only if you have created the HDR1 label with **BACULA.DATA** in the Filename field (starting with character 5). If Bacula writes the labels, it will use this information to recognize the tape as a Bacula tape. This allows ANSI/IBM labeled tapes to be used at sites with multiple machines and multiple backup programs.

41.1 Director Pool Directive

Label Type = ANSI | IBM | Bacula This directive is implemented in the Director Pool resource and in the SD Device resource. If it is specified in the SD Device resource, it will take precedence over the value passed from the Director to the SD. The default is Label Type = Bacula.

41.2 Storage Daemon Device Directives

Label Type = ANSI | IBM | Bacula This directive is implemented in the Director Pool resource and in the SD Device resource. If it is specified in the the SD Device resource, it will take precedence over the value passed from the Director to the SD.

Check Labels = <yes|no> This directive is implemented in the the SD Device resource. If you intend to read ANSI or IBM labels, this **must** be set. Even if the volume is not ANSI labeled, you can set this to yes, and Bacula will check the label type. Without this directive set to yes, Bacula will assume that labels are of Bacula type and will not check for ANSI or IBM labels. In other words, if there is a possibility of Bacula encountering an ANSI/IBM label, you must set this to yes.

Chapter 42

The Windows Version of Bacula

At the current time the File daemon or Client program has been thoroughly tested on Windows and is suitable for a production environment. A Windows version of the Bacula Storage daemon is also included in the installer, but it has not been extensively tested in a production environment. As a consequence, when we speak of the Windows version of Bacula below, we are referring to the File daemon (client) only.

The Windows version of the Bacula File daemon has been tested on WinXP, Windows 2000, Windows Server 2003, Windows Server 2008, Vista, Windows 7, Windows 8, Windows 10, and Windows 2012 systems. The Windows version of Bacula is a native Windows port, but there are very few source code changes to the Unix code, which means that the Windows version is for the most part running code that has long proved stable on Unix systems. When running, it is perfectly integrated with Windows. See section 42.1 for supported versions.

Once installed Bacula normally runs as a system service. This means that it is immediately started by the operating system when the system is booted, and runs in the background even if there is no user logged into the system.

42.1 Windows Supported Versions

It should be noted that as of 2016, the following Windows versions are no longer supported by Microsoft: Windows XP, Windows Vista, Windows 7, Windows Server 2000, and Windows Server 2003. As a consequence, these systems are no longer officially supported by Bacula. That said, we will make a best effort to support them, but cannot not guarantee that new Bacula Enterprise software developed after the support expiration date of each Microsoft OS will be supported.

42.2 Windows Installation

Normally, you will install the Windows version of Bacula from the binaries. This install is standard Windows .exe that runs an install wizard using the NSIS Free Software installer, so if you have already installed Windows software, it should be very familiar to you.

If you have a previous version of Bacula installed, you should stop the service, uninstall it, and remove the Bacula installation directory possibly saving your `bacula-fd.conf`, `bconsole.conf`, and `bat.conf` files for use with the new version you will install. The `Uninstall` program is normally found in `c:\textbackslash{}bacula\textbackslash{}Uninstall.exe`. We also recommend that you completely remove the directory `c:\bacula` because the current installer uses a different directory structure (see below).

Providing you do not already have Bacula installed, the installer installs the binaries and dlls in `c:\Program Files\Bacula\bin` and the configuration files in `c:\Documents and Settings\All Users\Application Data\Bacula` In addition, the **Start → All Programs → Bacula** menu item will be created during the installation, and on that menu, you will find items for editing the configuration files, displaying the document, and starting `bwx-console` or `bconsole`.

Finally, proceed with the installation.

- You must be logged in as Administrator to the local machine to do a correct installation, if not, please do so before continuing. Some users have attempted to install logged in as a domain administrator account and experienced permissions problems attempting to run Bacula, so we don't recommend that option.
- Simply double click on the `bacula-enterprise-win64-8.x.x.exe` NSIS install icon. The actual name of the icon will vary from one release version to another.

  bacula-enterprise-win64-8.x.x.exe
- Once launched, the installer wizard will ask you if you want to install Bacula.

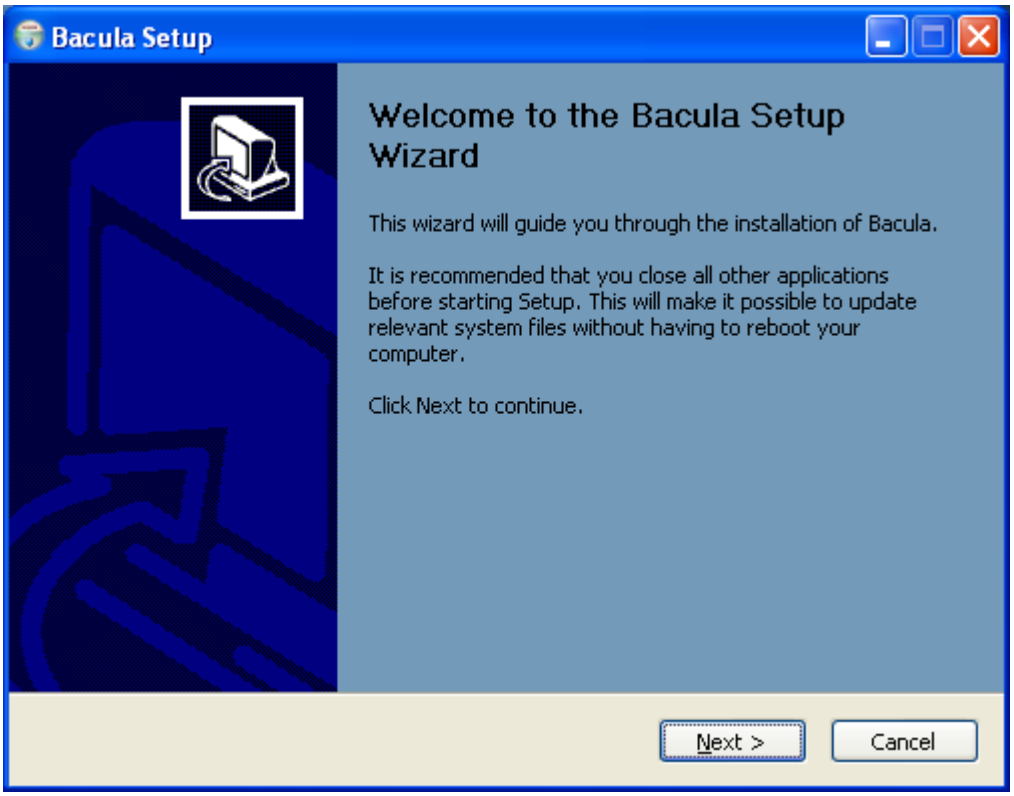


Figure 42.1: Windows Client Setup Wizard

- Next you will be asked to select the installation type.
- If you proceed, you will be asked to select the components to be installed. You may install the Bacula program (Bacula File Service) and or the documentation. Both will be installed in sub-directories of the install location that you choose later. The components dialog looks like the following:
- If you are installing for the first time, you will be asked to enter some very basic information about your configuration. If you are not sure what to enter, or have previously saved configuration files, you can put anything you want into the fields, then either replace the configuration files later with the ones saved, or edit the file.

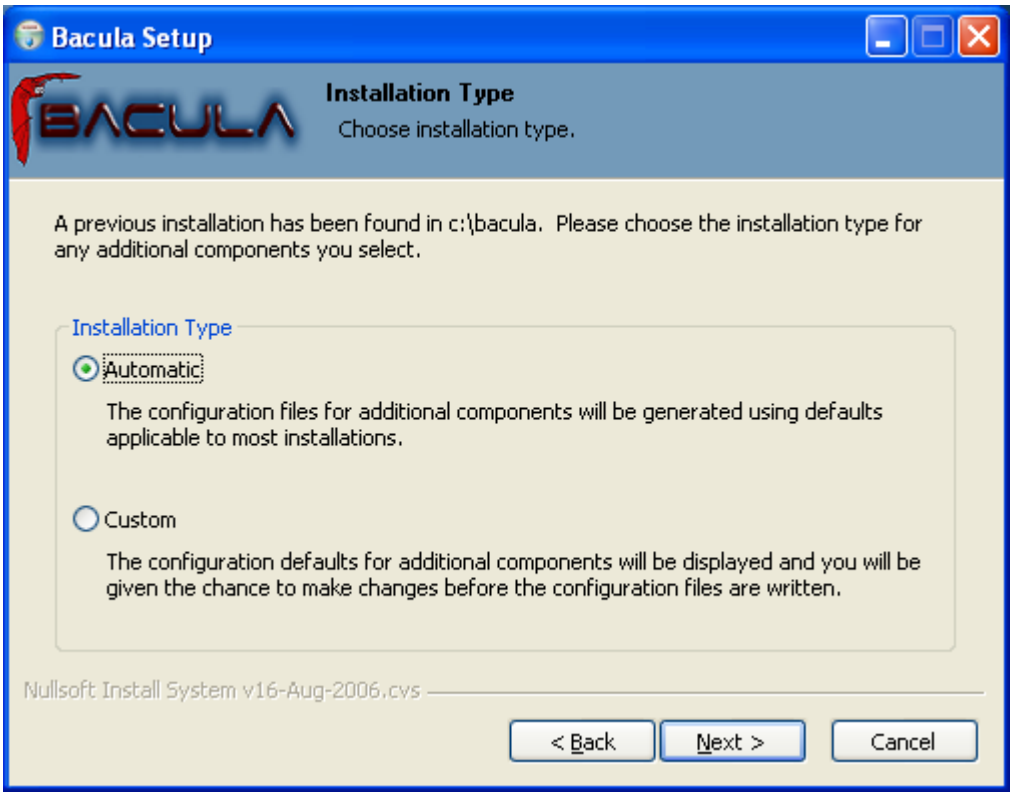


Figure 42.2: Windows Installation Type

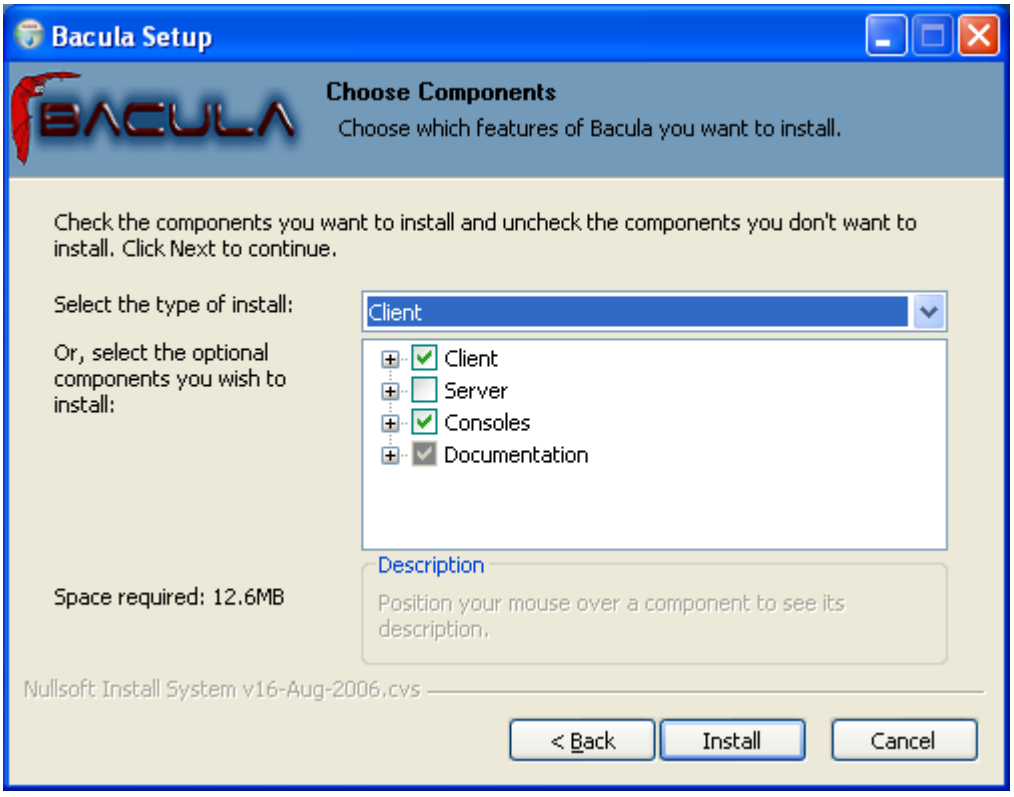


Figure 42.3: Win32 Component Selection Dialog

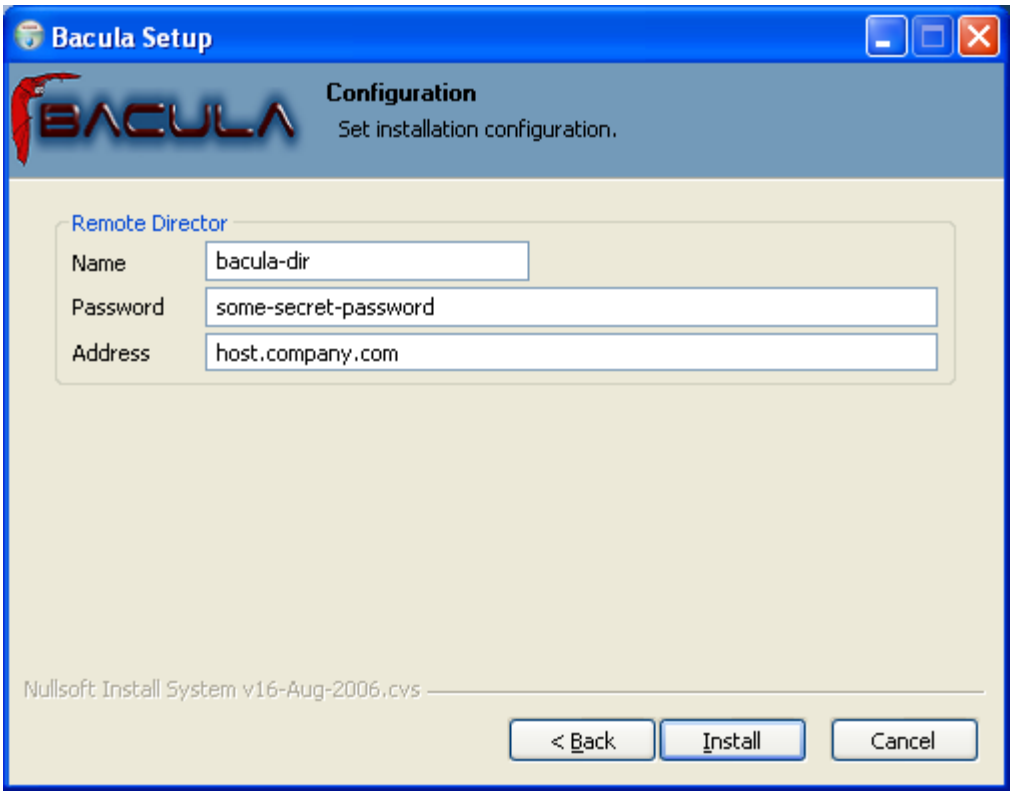


Figure 42.4: Win32 Configure

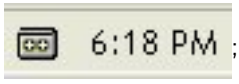
If you are upgrading an existing installation, the following will not be displayed.

- While the various files are being loaded, you will see the following dialog:
- Finally, the finish dialog will appear:

That should complete the installation process.

42.3 Tray Icon

Note: this section does not apply to all Windows versions later than Windows XP, since in those later versions Microsoft explicitly prohibits a system service such as Bacula from interacting with the user.

When the Bacula File Server is ready to serve files, an icon  representing a cassette (or tape) will appear in the system tray  ; right click on it and a menu will appear.

The **Events** item is currently unimplemented, by selecting the **Status** item, you can verify whether any jobs are running or not.

When the Bacula File Server begins saving files, the color of the holes in the cassette icon will change from white to green  , and if there is an error, the holes in the cassette icon will change to red  .

If you are using remote desktop connections between your Windows boxes, be warned that that

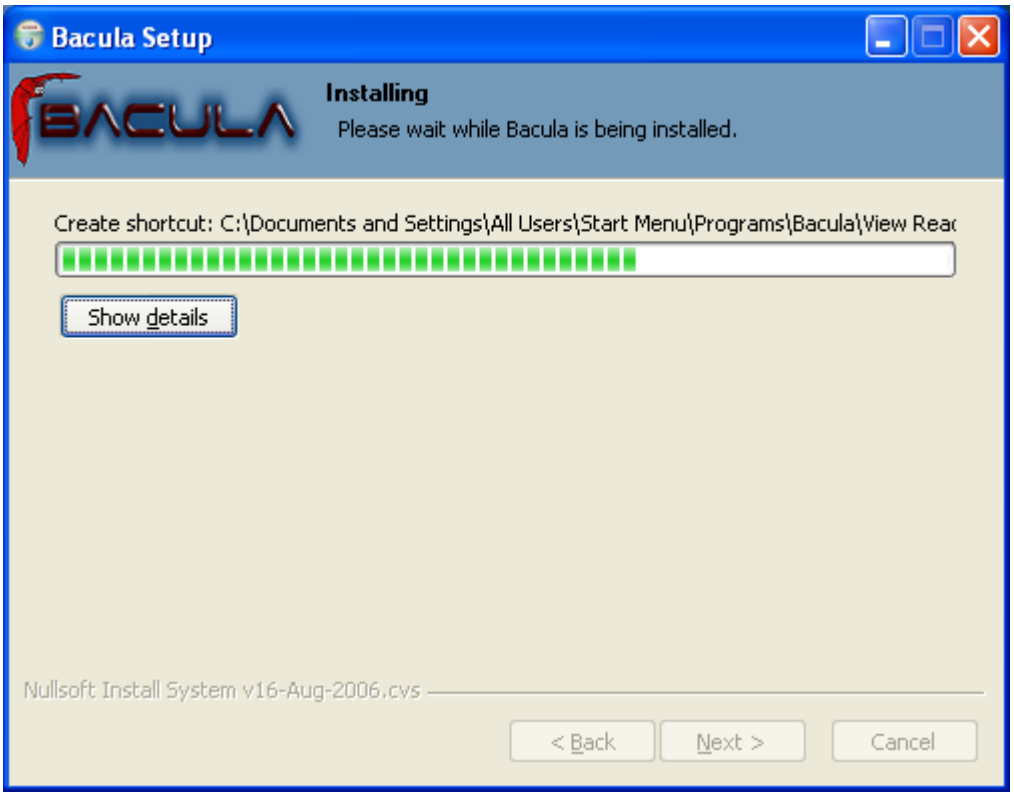


Figure 42.5: Windows Install Progress

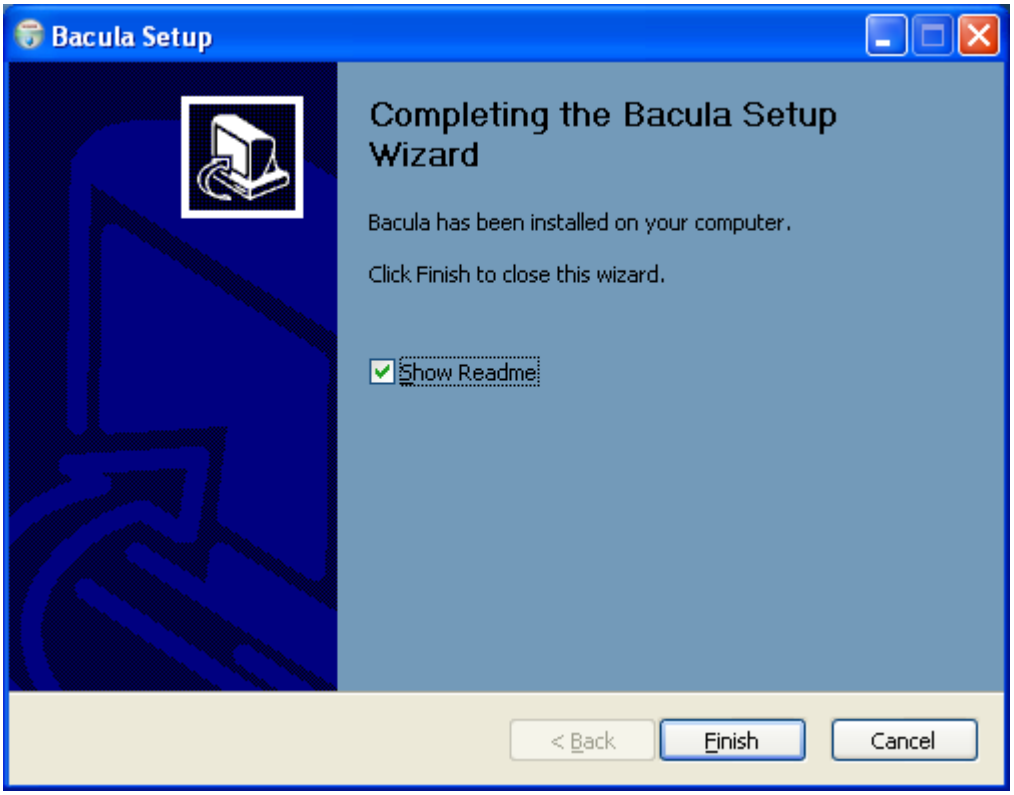


Figure 42.6: Windows Client Setup Completed

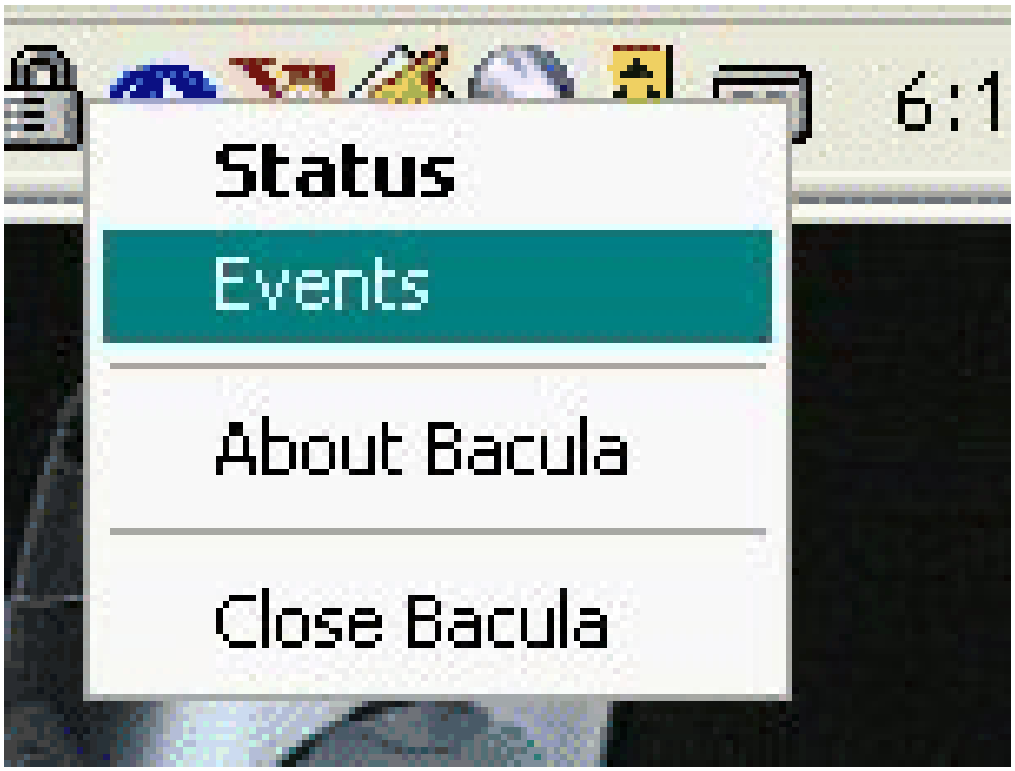


Figure 42.7: Menu on right click

tray icon does not always appear. It will always be visible when you log into the console, but the remote desktop may not display it.

42.4 Post Windows Installation

After installing Bacula and before running it, you should check the contents of the configuration files to ensure that they correspond to your installation. You can get to them by using: the **Start → All Programs → Bacula** menu item.

Finally, but pulling up the Task Manager (`CTRL-ALT-DEL`), verify that Bacula is running as a process (not an Application) with User Name SYSTEM. If this is not the case, you probably have not installed Bacula while running as Administrator, and hence it will be unlikely that Bacula can access all the system files.

42.5 Uninstalling Bacula on Windows

Once Bacula has been installed, it can be uninstalled using the standard Windows Add/Remove Programs dialog found on the Control panel.

42.6 Dealing with Windows Problems

Sometimes Windows machines the File daemon may have very slow backup transfer rates compared to other machines. To you might try setting the Maximum Network Buffer Size to 32,768

in both the File daemon and in the Storage daemon. The default size is larger, and apparently some Windows ethernet controllers do not deal with a larger network buffer size.

Many Windows ethernet drivers have a tendency to either run slowly due to old broken firmware, or because they are running in half-duplex mode. Please check with the ethernet card manufacturer for the latest firmware and use whatever techniques are necessary to ensure that the card is running in duplex.

If you are not using the portable option, and you have VSS (Volume Shadow Copy) enabled in the Director, and you experience problems with Bacula not being able to open files, it is most likely that you are running an antivirus program that blocks Bacula from doing certain operations. In this case, disable the antivirus program and try another backup. If it succeeds, either get a different (better) antivirus program or use something like RunClientJobBefore/After to turn off the antivirus program while the backup is running.

If turning off anti-virus software does not resolve your VSS problems, you might have to turn on VSS debugging. The following link describes how to do this: support.microsoft.com/kb/887013/en-us.

In Microsoft Windows Small Business Server 2003 the VSS Writer for Exchange is turned off by default. To turn it on, please see the following link: support.microsoft.com/default.aspx?scid=kb;EN-US;Q838183

The most likely source of problems is authentication when the Director attempts to connect to the File daemon that you installed. This can occur if the names and the passwords defined in the File daemon's configuration file `bacula-fd.conf` file on the Windows machine do not match with the names and the passwords in the Director's configuration file `bacula-dir.conf` located on your Unix/Linux server.

More specifically, the password found in the **Client** resource in the Director's configuration file must be the same as the password in the **Director** resource of the File daemon's configuration file. In addition, the name of the **Director** resource in the File daemon's configuration file must be the same as the name in the **Director** resource of the Director's configuration file.

It is a bit hard to explain in words, but if you understand that a Director normally has multiple Clients and a Client (or File daemon) may permit access by multiple Directors, you can see that the names and the passwords on both sides must match for proper authentication.

One user had serious problems with the configuration file until he realized that the Unix end of line conventions were used and Bacula wanted them in Windows format. This has not been confirmed though, and Bacula version 2.0.0 and above should now accept all end of line conventions (Windows, Unix, Mac).

Running Unix like programs on Windows machines is a bit frustrating because the Windows command line shell (DOS Window) is rather primitive. As a consequence, it is not generally possible to see the debug information and certain error messages that Bacula prints. With a bit of work, however, it is possible. When everything else fails and you want to **see** what is going on, try the following:

```
Start a DOS shell Window.
c:\Program Files\bacula\bacula-fd -t >out
type out
```

The precise path to bacula-fd depends on where it is installed. The **-t** option will cause Bacula to read the configuration file, print any error messages and then exit. the > redirects the output to the file named `out`, which you can list with the `type` command.

If something is going wrong later, or you want to run **Bacula** with a debug option, you might try starting it as:

```
c:\Program Files\bacula\bin\bacula-fd -d 100 >out
```

In this case, Bacula will run until you explicitly stop it, which will give you a chance to connect to it from your Unix/Linux server. When you start the File daemon in debug mode it can write the output to a trace file bacula.trace in the current directory. To enable this, before running a job, use the console, and enter:

```
trace on
```

then run the job, and once you have terminated the File daemon, you will find the debug output in the bacula.trace file, which will probably be located in the same directory as bacula-fd.exe.

In addition, you should look in the System Applications log on the Control Panel to find any Windows errors that Bacula got during the startup process.

Finally, due to the above problems, when you turn on debugging, and specify trace=1 on a setdebug command in the Console, Bacula will write the debug information to the file bacula.trace in the directory from which Bacula is executing.

If you are having problems with ClientRunBeforeJob scripts randomly dying, it is possible that you have run into an Oracle bug. See bug number 622 in the bugs.bacula.org database. The following information has been provided by a user on this issue:

```
The information in this document applies to:
 Oracle HTTP Server - Version: 9.0.4
 Microsoft Windows Server 2003
 Symptoms
 When starting an OC4J instance, the System Clock runs faster, about 7
seconds per minute.

 Cause

 + This is caused by the Sun JVM bug 4500388, which states that "Calling
Thread.sleep() with a small argument affects the system clock". Although
this is reported as fixed in JDK 1.4.0_02, several reports contradict this
(see the bug in
http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4500388).

 + Also reported by Microsoft as "The system clock may run fast when you
use the ACPI power management timer as a high-resolution counter on Windows
2000-based computers" (See http://support.microsoft.com/?id=821893)
```

You may wish to start the daemon with debug mode on rather than doing it using bconsole. To do so, edit the following registry key:

```
HKEY_LOCAL_MACHINE\HARDWARE\SYSTEM\CurrentControlSet\Services\Bacula-dir
```

using regedit, then add -dnn after the /service option, where nn represents the debug level you want.

42.7 Windows Compatibility Considerations

If you are not using the VSS (Volume Shadow Copy) option described in the next section of this chapter, and if any applications are running during the backup and they have files opened exclusively, Bacula will not be able to backup those files, so be sure you close your applications (or tell your users to close their applications) before the backup. Fortunately, most Microsoft applications do not open files exclusively so that they can be backed up. However, you will need

to experiment. In any case, if Bacula cannot open the file, it will print an error message, so you will always know which files were not backed up. See the section below on Volume Shadow Copy Service that permits backing up any file.

During backup, Bacula doesn't know about the system registry, so you will either need to write it out to an ASCII file using **regedit /e** or use a program specifically designed to make a copy or backup the registry.

Bacula uses Windows backup API calls by default. Typical of Windows, programming these special BackupRead and BackupWrite calls is a real nightmare of complications. The end result gives some distinct advantages and some disadvantages.

First, the advantages are that Windows systems, the security and ownership information is now backed up. In addition, with the exception of files in exclusive use by another program, Bacula can now access all system files. This means that when you restore files, the security and ownership information will be restored on Windows along with the data.

The disadvantage of the Windows backup API calls is that it produces non-portable backups. That is files and their data that are backed up on Windows using the native API calls (BackupRead/BackupWrite) cannot be directly restored on Linux or Unix systems. Bacula should be able to read non-portable backups on any system and restore the data appropriately. However, on a system that does not have the BackupRead/BackupWrite calls (older Windows versions and all Unix/Linux machines), though the file data can be restored, the Windows security and access control data will not be restored. This means that a standard set of access permissions will be set for such restored files.

As a default, Bacula backs up Windows systems using the Windows API calls. If you want to backup data on a Windows system and restore it on a Unix or Linux system, we have provided a special **portable** option that backs up the data in a portable fashion by using portable API calls. See the portable option on the Include statement in a FileSet resource in the Director's configuration chapter for the details on setting this option. However, using the portable option means you may have permissions problems accessing files, and none of the security and ownership information will be backed up or restored. The file data can, however, be restored on any system.

You should always be able to restore any file backed up on Unix or Win95/98/Me to any other system. On some older Windows systems, you may have to reset the ownership of such restored files.

Finally, if you specify the **portable=yes** option on the files you back up. Bacula will be able to restore them on any other system. However, any Windows specific security and ownership information will be lost.

The following matrix will give you an idea of what you can expect. Thanks to Marc Brueckner for doing the tests:

Table 42.1: WinNT/2K/XP Restore Portability Status

Backup OS	Restore OS	Results
Win Me		
Win Me	Win Me	Works
Win Me	Win NT	Works (SYSTEM permissions)
Win Me	Win XP	Works (SYSTEM permissions)
Win Me	Linux	Works (SYSTEM permissions)
Win XP		
Win XP	Win XP	Works

Continues on the following page

[]Cont.

Backup OS	Restore OS	Results
Win XP	Win NT	Works (all files OK, but got "The data is invalid" message)
Win XP	Win Me	Error: Win32 data stream not supported.
Win XP	Win Me	Works if Portable=yes specified during backup.
Win XP	Linux	Error: Win32 data stream not supported.
Win XP	Linux	Works if Portable=yes specified during backup.
Win NT		
Win NT	Win NT	Works
Win NT	Win XP	Works
Win NT	Win Me	Error: Win32 data stream not supported.
Win NT	Win Me	Works if Portable=yes specified during backup.
Win NT	Linux	Error: Win32 data stream not supported.
Win NT	Linux	Works if Portable=yes specified during backup.
Linux		
Linux	Linux	Works
Linux	Win NT	Works (SYSTEM permissions)
Linux	Win Me	Works
Linux	Win XP	Works (SYSTEM permissions)

Note: Non-portable Windows data can be restore to any machine. In addition, with recent Bacula versions, Bacula is able to extract data part from Windows backup streams when restoring on a non-Windows machine.

42.8 Volume Shadow Copy Service

Microsoft added VSS to Windows XP and Windows 2003. From the perspective of a backup-solution for Windows, this is an extremely important step. VSS allows Bacula to backup open files and even to interact with applications like RDBMS to produce consistent file copies. VSS aware applications are called VSS Writers, they register with the OS so that when Bacula wants to do a Snapshot, the OS will notify the register Writer programs, which may then create a consistent state in their application, which will be backed up. Examples for these writers are "MSDE" (Microsoft Database Engine), "Event Log Writer", "Registry Writer" plus 3rd party-writers. If you have a non-vss aware application (e.g. SQL Anywhere or probably MySQL), a shadow copy is still generated and the open files can be backed up, but there is no guarantee that the file is consistent.

Bacula produces a message from each of the registered writer programs when it is doing a VSS backup so you know which ones are correctly backed up.

Technically Bacula creates a shadow copy as soon as the backup process starts. It does then backup all files from the shadow copy and destroys the shadow copy after the backup process. Please have in mind, that VSS creates a snapshot and thus backs up the system at the state it had when starting the backup. It will disregard file changes which occur during the backup process.

VSS can be turned on by placing an

```
Enable VSS = yes
```

in your FileSet resource.

The VSS aware File daemon has the letters VSS on the signon line that it produces when contacted by the console. For example:

```
Tibs-fd Version: 1.37.32 (22 July 2005) VSS Windows XP MVS NT 5.1.2600
```

the VSS is shown in the line above. This only means that the File daemon is capable of doing VSS not that VSS is turned on for a particular backup. There are two ways of telling if VSS is actually turned on during a backup. The first is to look at the status output for a job, e.g.:

```
Running Jobs:
JobId 1 Job NightlySave.2005-07-23_13.25.45 is running.
    VSS Backup Job started: 23-Jul-05 13:25
    Files=70,113 Bytes=3,987,180,650 Bytes/sec=3,244,247
    Files Examined=75,021
    Processing file: c:/Documents and Settings/kern/My Documents/My Pictures/Misc1/Sans titre - 39.pdd
    SDReadSeqNo=5 fd=352
```

Here, you see under Running Jobs that JobId 1 is "VSS Backup Job started ..." This means that VSS is enabled for that job. If VSS is not enabled, it will simply show "Backup Job started ..." without the letters VSS.

The second way to know that the job was backed up with VSS is to look at the Job Report, which will look something like the following:

```
23-Jul 13:25 rufus-dir: Start Backup JobId 1, Job=NightlySave.2005-07-23_13.25.45
23-Jul 13:26 rufus-sd: Wrote label to prelabeled Volume "TestVolume001" on device "DDS-4" (/dev/nst0)
23-Jul 13:26 rufus-sd: Spooling data ...
23-Jul 13:26 Tibs: Generate VSS snapshots. Driver="VSS WinXP", Drive(s)="C"
23-Jul 13:26 Tibs: VSS Writer: "MSDEWriter", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "Microsoft Writer (Bootable State)", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "WMI Writer", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "Microsoft Writer (Service State)", State: 1 (VSS_WS_STABLE)
```

In the above Job Report listing, you see that the VSS snapshot was generated for drive C (if other drives are backed up, they will be listed on the **Drive(s)="C"** You also see the reports from each of the writer program. Here they all report VSS_WS_STABLE, which means that you will get a consistent snapshot of the data handled by that writer.

42.9 VSS Problems

If you are experiencing problems such as VSS hanging on MSDE, first try running `vssadmin` to check for problems, then try running `ntbackup` which also uses VSS to see if it has similar problems. If so, you know that the problem is in your Windows machine and not with Bacula.

The FD hang problems were reported with **MSDEwriter** when:

- a local firewall locked local access to the MSDE TCP port (MSDEwriter seems to use TCP/IP and not Named Pipes).
- `msdtcs` was installed to run under "localsystem": try running `msdtcs` under networking account (instead of local system) (com+ seems to work better with this configuration).

42.10 Windows Firewalls

If you turn on the firewalling feature on Windows (default in WinXP SP2), you are likely to find that the Bacula ports are blocked and you cannot communicate to the other daemons. This can be deactivated through the **Security Notification** dialog, which is apparently somewhere in the **Security Center**. I don't have this on my computer, so I cannot give the exact details.

The command:

```
netsh firewall set opmode disable
```

is purported to disable the firewall, but this command is not accepted on my WinXP Home machine.

42.11 Windows Port Usage

If you want to see if the File daemon has properly opened the port and is listening, you can enter the following command in a shell window:

```
netstat -an | findstr 910[123]
```

TopView is another program that has been recommend, but it is not a standard Windows program, so you must find and download it from the Internet.

42.12 Windows Disaster Recovery

Bacula Systems provides a special WinBMR (Windows Bare Metal Recovery) program that allows you to perform a bare metal disaster recovery of your system.

If you do not have the WinBMR program, we do not have a good solution for disaster recovery on Windows. The main piece lacking, which is available in the Bacula Enterprise WinBMR, is a Windows boot floppy or a Windows boot CD. Microsoft releases a Windows Pre-installation Environment (**WinPE**) that could possibly work, but we have not investigated it. This means that until someone figures out the correct procedure, you must restore the OS from the installation disks, then you can load a Bacula client and restore files. Please don't count on using bextract to extract files from your backup tapes during a disaster recovery unless you have backed up those files using the **portable** option. bextract does not run on Windows, and the normal way Bacula saves files using the Windows API prevents the files from being restored on a Unix machine. Once you have an operational Windows OS loaded, you can run the File daemon and restore your user files.

42.13 Windows FD Restrictions

In recent versions of Windows, Microsoft has implemented Volume Mount Points, Encrypted Volumes, and Deduplicated Volumes. Current versions of the Windows File daemon support Volume Mount Points much like on Linux systems. That is you must explicitly add a File= line in your FileSet so that it will decend into the mount point. In addition, during the restore of a mount point, the mount point must already be defined.

Current Bacula File daemons do not support Encrypted Volumes or Deduplicated Volumes.

42.14 Windows Restore Problems

Please see the Restore Chapter of this manual for problems that you might encounter doing a restore.

sectionWindows Backup Problems If during a Backup, you get the message: **ERR=Access is denied** and you are using the portable option, you should try both adding both the non-portable (backup API) and the Volume Shadow Copy options to your Director's conf file.

In the Options resource:

```
portable = no
```

In the FileSet resource:

```
EnableVSS = yes
```

In general, specifying these two options should allow you to backup any file on a Windows system. However, in some cases, if users have allowed to have full control of their folders, even system programs such a Bacula can be locked out. In this case, you must identify which folders or files are creating the problem and do the following:

1. Grant ownership of the file/folder to the Administrators group, with the option to replace the owner on all child objects.
2. Grant full control permissions to the Administrators group, and change the user's group to only have Modify permission to the file/folder and all child objects.

Thanks to Georger Araujo for the above information.

42.15 Windows Ownership and Permissions Problems

If you restore files backed up from Windows to an alternate directory, Bacula may need to create some higher level directories that were not saved (or restored). In this case, the File daemon will create them under the SYSTEM account because that is the account that Bacula runs under as a service. As of version 1.32f-3, Bacula creates these files with full access permission. However, there may be cases where you have problems accessing those files even if you run as administrator. In principle, Microsoft supplies you with the way to cease the ownership of those files and thus change the permissions. However, a much better solution to working with and changing Windows permissions is the program `SetACL`, which can be found at setacl.sourceforge.net.

If you have not installed Bacula while running as Administrator and if Bacula is not running as a Process with the userid (User Name) SYSTEM, then it is very unlikely that it will have sufficient permission to access all your files.

Some users have experienced problems restoring files that participate in the Active Directory. They also report that changing the userid under which Bacula (`bacula-fd.exe`) runs, from SYSTEM to a Domain Admin userid, resolves the problem.

42.16 Manually resetting the Permissions

The following solution was provided by Dan Langille <dan at langille in the dot org domain>. The steps are performed using Windows 2000 Server but they should apply to most Windows

platforms. The procedure outlines how to deal with a problem which arises when a restore creates a top-level new directory. In this example, "top-level" means something like `c:\src`, not `c:\tmp\src` where `c:\tmp` already exists. If a restore job specifies `/` as the **Where:** value, this problem will arise.

The problem appears as a directory which cannot be browsed with Windows Explorer. The symptoms include the following message when you try to click on that directory:

Figure 42.8: Popup on permission issue

If you encounter this message, the following steps will change the permissions to allow full access.

1. right click on the top level directory (in this example, `c:/src`) and select **Properties**.
2. click on the Security tab.
3. If the following message appears, you can ignore it, and click on **OK**.

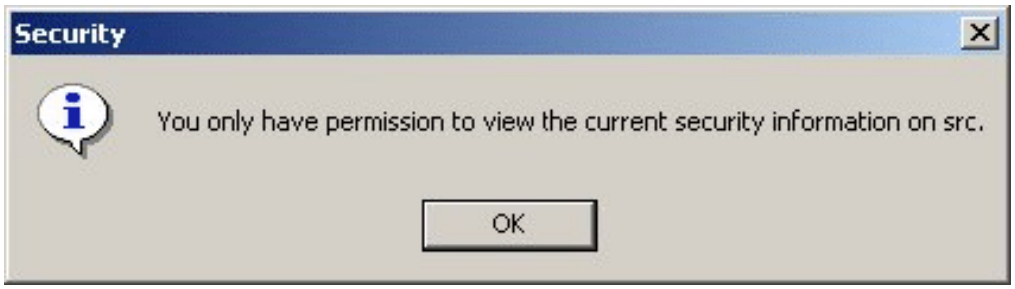


Figure 42.9: Message to ignore

You should see something like this:

4. click on Advanced
5. click on the Owner tab
6. Change the owner to something other than the current owner (which is **SYSTEM** in this example as shown below).
7. ensure the "Replace owner on subcontainers and objects" box is checked
8. click on OK
9. When the message "You do not have permission to read the contents of directory `c:\src` basis. Do you wish to replace the directory permissions with permissions granting you Full Control?", click on Yes.
10. Click on OK to close the Properties tab

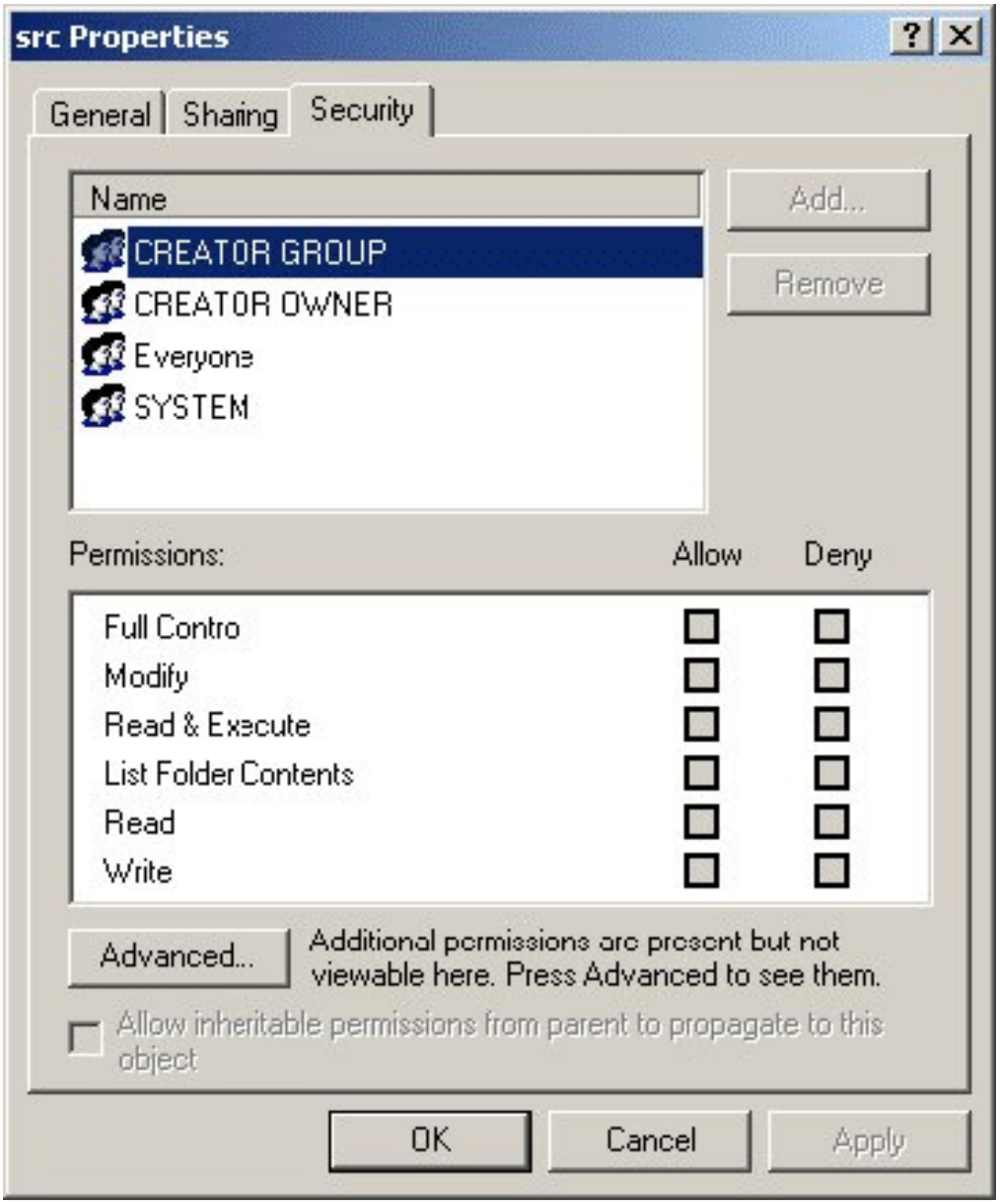


Figure 42.10: Properties security

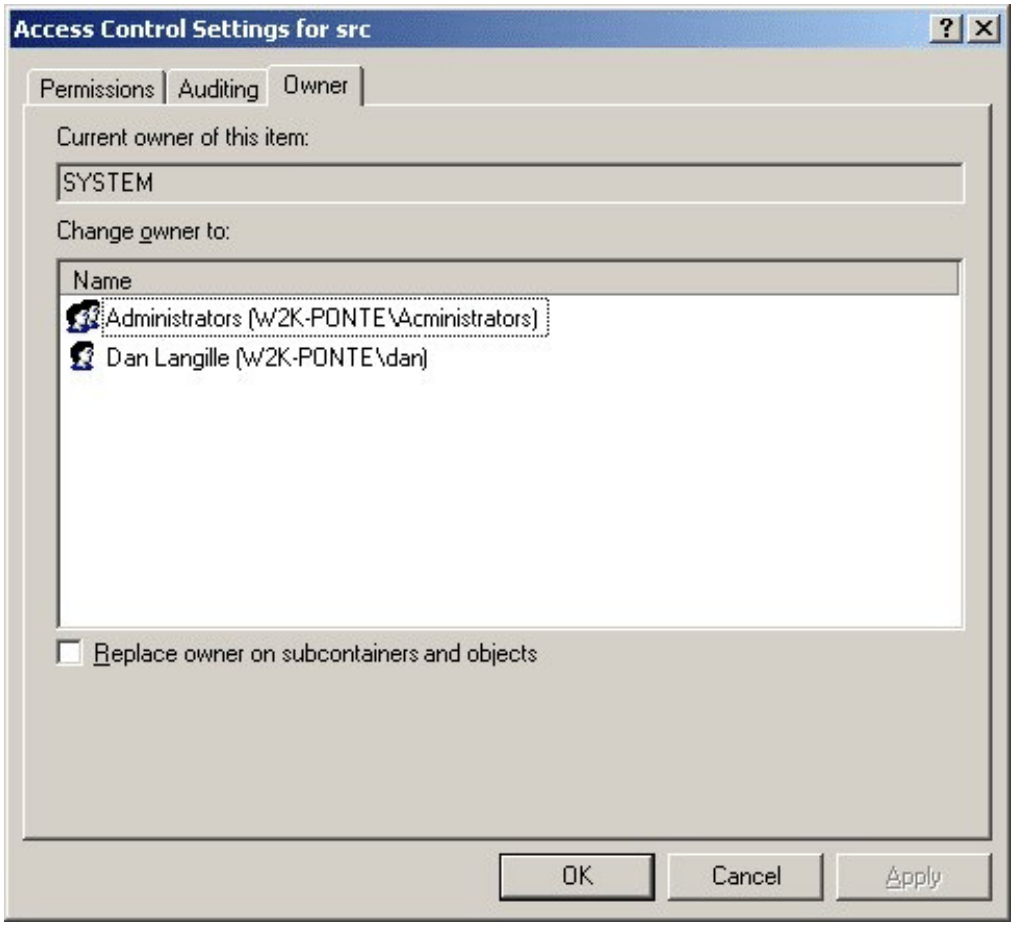


Figure 42.11: Properties security advanced owner

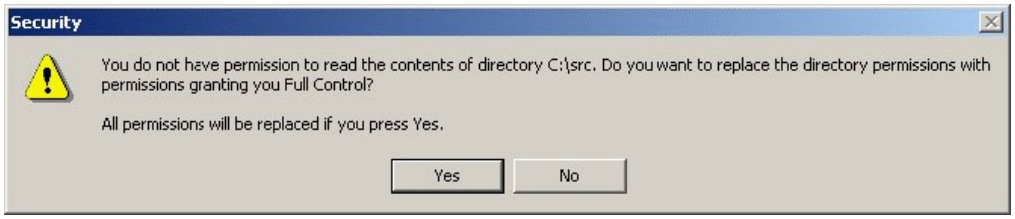


Figure 42.12: Confirm granting permissions

With the above procedure, you should now have full control over your restored directory.

In addition to the above methods of changing permissions, there is a Microsoft program named cacls that can perform similar functions.

42.17 Backing Up the WinNT/XP/2K System State

Note, most of this section applies to the older Windows OSes that do not have VSS. On newer Windows OSes that have VSS, all files including the System State will by default be properly backed up by Bacula. In addition, if you want special Backup of the System State (rather than just backup of everything) or backup of MSSQL or MS Exchange, you should consider the Bacula Systems VSS plugin (not to be confused with enabling VSS snapshots in the FileSet). The VSS plugin has explicit knowledge that allow you to make special backups of Microsoft System components that have VSS writers. Bacula Systems provides specific white papers on the components that are available for the VSS plugin.

If you want to do this on your own, a suggestion by Damian Coutts using Microsoft's NTBackup utility in conjunction with Bacula should permit a full restore of any damaged system files on Win2K/XP. His suggestion is to do an NTBackup of the critical system state prior to running a Bacula backup with the following command:

```
ntbackup backup systemstate /F c:\systemstate.bkf
```

The **backup** is the command, the **systemstate** says to backup only the system state and not all the user files, and the **/F c:\systemstate.bkf** specifies where to write the state file. this file must then be saved and restored by Bacula.

To restore the system state, you first reload a base operating system if the OS is damaged, otherwise, this is not necessary, then you would use Bacula to restore all the damaged or lost user's files and to recover the c:\textbackslash{}systemstate.bkf file. Finally if there are any damaged or missing system files or registry problems, you run NTBackup and **catalogue** the system statefile, and then select it for restore. The documentation says you can't run a command line restore of the systemstate.

To the best of my knowledge, this has not yet been tested. If you test it, please report your results to the Bacula email list.

Note, Bacula uses VSS to backup and restore open files and system files, but on older Windows machines such as WinNT and Win2000, VSS is not implemented by Microsoft so that you must use some special techniques to back them up as described above. On new Windows machines, Bacula will backup and restore all files including the system state providing you have VSS enabled in your Bacula FileSet (default).

42.18 Fixing the Windows Boot Record

A tip from a user: An effective way to restore a Windows backup for those who do not purchase the bare metal restore capability is to install Windows on a different hard drive and restore the backup. Then run the recovery CD and run

```
diskpart
   select disk 0
   select part 1
   active
   exit

bootrec /rebuldbcd
```

```
bootrec /fixboot
bootrec /fixmbr
```

42.19 Considerations for Filename Specifications

Please see the Director's Configuration chapter of this manual for important considerations on how to specify Windows paths in Bacula FileSet Include and Exclude directives.

The Window Bacula File daemon as well as bconsole and bwx-console support Windows Unicode filenames. There may still be some problems with multiple byte characters (e.g. Chinese, ...) where it is a two byte character but the displayed character is not two characters wide.

Path/filenames longer than 260 characters (up to 32,000) are supported.

42.20 Windows Specific File daemon Command Line

These options are not normally seen or used by the user, and are documented here only for information purposes. At the current time, to change the default options, you must either manually run **Bacula** or you must manually edit the system registry and modify the appropriate entries.

In order to avoid option clashes between the options necessary for **Bacula** to run on Windows and the standard Bacula options, all Windows specific options are signaled with a forward slash character (/), while as usual, the standard Bacula options are signaled with a minus (-), or a minus minus (--). All the standard Bacula options can be used on the Windows version. In addition, the following Windows only options are implemented:

/service Start Bacula as a service

/run Run the Bacula application

/install Install Bacula as a service in the system registry

/remove Uninstall Bacula from the system registry

/about Show the Bacula about dialogue box

/status Show the Bacula status dialogue box

/events Show the Bacula events dialogue box (not yet implemented)

/kill Stop any running **Bacula**

/help Show the Bacula help dialogue box

It is important to note that under normal circumstances the user should never need to use these options as they are normally handled by the system automatically once Bacula is installed. However, you may note these options in some of the .bat files that have been created for your use.

42.21 Shutting down Windows Systems

Some users like to shutdown their Windows machines after a backup using a Client Run After Job directive. If you want to do something similar, you might take the shutdown program from

the apcupsd project[1] or one from the Sysinternals project[2].

[1] http://www.apcupsd.com
[2] http://technet.microsoft.com/en-us/sysinternals/bb897541.aspx

Chapter 43

Disaster Recovery Using Bacula

43.1 General

When disaster strikes, you must have a plan, and you must have prepared in advance otherwise the work of recovering your system and your files will be considerably greater. For example, if you have not previously saved the partitioning information for your hard disk, how can you properly rebuild it if the disk must be replaced? Unfortunately, many of the steps one must take before and immediately after a disaster are very operating system dependent.

Bacula Systems provides special LinuxBMR and WinBMRpackages that largely automate and simplify recovery even with a very large number of machines. The documentation and details of these products are contained in separate white papers. If you need them, please ask your Bacula Systems sales representative.

Below are a few additional points that may or may not help.

43.2 Important Considerations

Here are a few important considerations concerning disaster recovery that you should take into account before a disaster strikes.

- If the building which houses your computers burns down or is otherwise destroyed, do you have off-site backup data?
- Disaster recovery is much easier if you have several machines. If you have a single machine, how will you handle unforeseen events if your only machine is down?
- Do you want to protect your whole system and use Bacula to recover everything? or do you want to try to restore your system from the original installation disks and apply any other updates and only restore user files?

43.3 FreeBSD Bare Metal Recovery

The same basic techniques described above also apply to FreeBSD. Although we don't yet have a fully automated procedure, Alex Torres Molina has provided us with the following instructions with a few additions from Jesse Guardiani and Dan Langille:

1. Boot with the FreeBSD installation disk

2. Go to Custom, Partition and create your slices and go to Label and create the partitions that you want. Apply changes.

3. Go to Fixit to start an emergency console.

4. Create devs ad0 .. .. if they don't exist under /mnt2/dev (in my situation) with MAKEDEV. The device or devices you create depend on what hard drives you have. ad0 is your first ATA drive. da0 would by your firstSCSI drive. Under OS version 5 and greater, your device files are most likely automatically created for you.

5. ```
   mkdir /mnt/disk
   ```

   this is the root of the new disk

6. ```
   mount /mnt2/dev/ad0s1a /mnt/disk
   mount /mnt2/dev/ad0s1c /mnt/disk/var
   mount /mnt2/dev/ad0s1d /mnt/disk/usr
   .....
   ```

   The same hard drive issues as above apply here too. Note, under OS version 5 or higher, your disk devices may be in /dev not /mnt2/dev.

7. Network configuration

   ```
   ifconfig xl0 ip/mask + route add default ip-gateway
   ```

8. ```
   mkdir /mnt/disk/tmp
   ```

9. ```
   cd /mnt/disk/tmp
   ```

10. Copy bacula-fd and bacula-fd.conf to this path

11. If you need to, use sftp to copy files, after which you must do this:

    ```
    ln -s /mnt2/usr/bin /usr/bin
    ```

12. ```
    chmod u+x bacula-fd
    ```

13. Modify bacula-fd.conf to fit this machine

14. Copy /bin/sh to /mnt/disk, necessary for chroot

15. Don't forget to put your bacula-dir's IP address and domain name in /mnt/disk/etc/hosts if it's not on a public net. Otherwise the FD on the machine you are restoring to won't be able to contact the SD and DIR on the remote machine.

16. ```
    mkdir -p /mnt/disk/var/db/bacula
    ```

17. ```
    chroot /mnt/disk /tmp/bacula-fd -c /tmp/bacula-fd.conf
    ```

    to start bacula-fd

18. Now you can go to bacula-dir and restore the job with the entire contents of the broken server.

19. You must create /proc
    ```

43.4 Solaris Bare Metal Recovery

The same basic techniques described above apply to Solaris:

- the same restrictions as those given for Linux apply
- you will need to create a Rescue disk

However, during the recovery phase, the boot and disk preparation procedures are different:

- there is no need to create an emergency boot disk since it is an integrated part of the Solaris boot.
- you must partition and format your hard disk by hand following manual procedures as described in W. Curtis Preston's book "Unix Backup & Recovery"

Once the disk is partitioned, formatted and mounted, you can continue with bringing up the network and reloading Bacula.

43.5 Preparing Solaris Before a Disaster

As mentioned above, before a disaster strikes, you should prepare the information needed in the case of problems. To do so, in the `rescue/solaris` subdirectory enter:

```
su
./getdiskinfo
./make_rescue_disk
```

The `getdiskinfo` script will, as in the case of Linux described above, create a subdirectory `diskinfo` containing the output from several system utilities. In addition, it will contain the output from the `SysAudit` program as described in Curtis Preston's book. This file `diskinfo/sysaudit.bsi` will contain the disk partitioning information that will allow you to manually follow the procedures in the "Unix Backup & Recovery" book to repartition and format your hard disk. In addition, the `getdiskinfo` script will create a `start_network` script.

Once you have your disks repartitioned and formatted, do the following:

- Start Your Network with the `start_network` script
- Restore the Bacula File daemon as documented above
- Perform a Bacula restore of all your files using the same commands as described above for Linux
- Re-install your boot loader using the instructions outlined in the "Unix Backup & Recovery" book using installboot

Chapter 44

Bacula TLS – Communications Encryption

Bacula Transport Layer Security (TLS) is built-in network encryption code to provide secure network transport similar to that offered by `stunnel` or `ssh`. The Bacula TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code. For data encryption, please see the PKI options described in Data Encryption chapter of this manual.

The Bacula encryption implementation was initially written by Landon Fuller.

Supported features of this code include:

- Client/Server TLS Requirement Negotiation
- TLSv1.2 Connections with Server and Client Certificate Validation
- Forward Secrecy Support via Diffie-Hellman Ephemeral Keying
- TLS-PSK is used by default[1] when TLS certificates are not configured.

This document will refer to both "server" and "client" contexts. These terms refer to the accepting and initiating peer, respectively. In addition, each of the three daemons (Director, File daemon, Storage daemon) as well as the user interface programs (`bconsole`, `tray monitor`, ...) use the same TLS configuration directives. When we are speaking of one or all of these daemons/programs, we will generally refer to them as a "component".

Diffie-Hellman anonymous ciphers are not supported by this code. The use of DH anonymous ciphers increases the code complexity and places explicit trust upon the two-way CRAM-MD5 implementation. CRAM-MD5 is subject to known plaintext attacks, and it should be considered considerably less secure than PKI certificate-based authentication.

Appropriate autoconf macros have been added to detect and use OpenSSL[2] if enabled on the `./configure` line with **--with-openssl**

44.1 TLS Configuration Directives

Additional configuration directives have been added to all the components (daemons) (Director, File daemon, and Storage daemon) as well as the various different Console programs).

[1] starting with version 12.0
[2] https://www.openssl.org/

Note that for the connection between a Storage Daemon and a File Daemon or between two Storage Daemons, the TLS directives that will be used are the ones defined in the Client `FileDaemon` resource and in the Storage `Storage` resource.

The default value of the directive **TLS-PSK Enable** is yes, if both **TLS Enable** and **TLS-PSK Enable** are enable on both side, then Bacula will use TLS certificates.

If none of TLS or TLS-PSK are enabled, then the TLS directives have no effect.

These new directives are defined as follows:

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all Bacula components. The Pre-Shared Key used between the programs is the Bacula password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = <yes|no> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to yes. If TLS is not required while TLS or TLS-PSK are enabled, then the Bacula component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the Bacula component will refuse any connection request that does not use TLS.

TLS Authenticate = <yes|no> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to no in the corresponding server context.

Example:

File Daemon configuration file (`bacula-fd.conf`), `Director` resource configuration has **TLS Verify Peer**=no:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com

  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = no
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer**=no, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the `Client` resource, director configuration file. We can have the below client configuration in `bacula-dir.conf`:

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509[3] certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowed CN** onfiguration directive is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is yes.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to no (**TLS Verify Peer** is yes by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in `bacula-fd.conf`, `Director` resource definition:

```
Director {
  Name = bacula-dir
  Password = "password"
  Address = director.example.com
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
  TLS Verify Peer = yes
  TLS Allowed CN = director.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
  TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
  Name = client1-fd
  Address = client1.example.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "password"
  ...
  # TLS configuration directives
  TLS Enable = yes
  TLS Require = yes
  # the Allowed CN will be checked for this client by director
  # the client's certificate Common Name must match any of
  # the values of the Allowed CN list
  TLS Allowed CN = client1.example.com
  TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
  TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
```

[3] https://en.wikipedia.org/wiki/X.509

```
    TLS Key = /opt/bacula/ssl/keys/director_key.pem
  }
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to no, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to no, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use openssl:

```
openssl dhparam -out dh4096.pem -5 4096
```

44.2 Creating a Self-Signed Certificate

A self-signed certificate for use with the Bacula TLS that will work, but will not allow proper certificate validation can be created easily. The .pem file containing both the certificate and the key valid for about ten years can be created as follows:

```
openssl req -new -x509 -nodes -out bacula.pem -keyout bacula.pem -days 3650
```

The above command will ask a number of questions. You may simply answer each of them by entering a return, or if you wish you may enter your own data.

Note, however, that self-signed certificates will only work for the outgoing end of connections. For example, in the case of the Director making a connection to a File Daemon, the File Daemon may be configured to allow self-signed certificates, but the certificate used by the Director must be signed by a certificate that is explicitly trusted on the File Daemon end.

This is necessary to prevent "man in the middle" attacks from tools such as ettercap[4]. Essentially, if the Director does not verify that it is talking to a trusted remote endpoint, it can be tricked into talking to any malicious 3^{rd} party who is relaying and capturing all traffic by presenting its own certificates to the Director and File Daemons. The only way to prevent this is by using trusted certificates, so that the man in the middle is incapable of spoofing the connection using his own.

[4] https://www.ettercap-project.org/

44.3 Getting a CA Signed Certificate

The process of getting a certificate that is signed by a CA is quite a bit more complicated. You can purchase one from a number of PKI vendors, but that is not at all necessary for use with Bacula.

To get a CA signed certificate, you will either need to find a friend that has setup his own, trusted CA, or to become a CA yourself, and thus you can sign all your own certificates. The book OpenSSL by John Viega, Matt Mesier & Pravir Chandra from O'Reilly[56] explains how to do it, or you can read the documentation provided in the Open-source PKI Book[7].

44.4 Example TLS Configuration Files

A few examples of the TLS portions of configuration files are shown, which should help you setting up your own.

For the examples in this section, we will consider three hosts named darkstar, arrakis and caladan, in order to have examples with components running in different machines. The following private key and public certificate files will be used:

```
* host darkstar.example.com:

/opt/bacula/ssl/keys/darkstar_key.pem
/opt/bacula/ssl//certs/darkstar_cert.pem

* host arrakis.example.com:

/opt/bacula/ssl/keys/arrakis_key.pem
/opt/bacula/ssl//certs/arrakis_cert.pem

* host caladan.example.com:

/opt/bacula/ssl/keys/caladan_key.pem
/opt/bacula/ssl//certs/caladan_cert.pem
```

The **TLS Verify Peer = yes** is present in the below examples to emphasize where it can be configured, but there is no need since this is the **default value used by Bacula**. This means that both server and client certificates will be checked in the TLS communication handshake. The Common Name (CN) in the peer certificate will be checked against the **Address** directive configured for the corresponding resource.

The **TLS Allowed CN = <FQDN! or IP address>** is configured to ensure that only the peer certificates with a CN Subject field listed here are authorized to communicate. For example, consider the below certificate:

```
# openssl x509 -in ../ssl/certs/arrakis_cert.pem -text -noout
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 1 (0x1)
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=US, ST=Yverdon-les-Bains, L=Vaud, O=Bacula Systems, CN=ca.baculasystems.com/emailAddress=example@
Validity
  Not Before: Sep 7 09:50:27 2016 GMT
  Not After : Dec 31 23:59:00 2021 GMT
Subject: C=CH, ST=Madrid, O=BaculaSystems, CN=arrakis.example.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
```

[5] shop.oreilly.com/product/9780596002701.do
[6] This link is valid as of the 27-september-2018
[7] http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm

```
Public-Key: (4096 bit)
Modulus:
    00:ae:7f:3f:6f:22:27:c3:c5:f5:d3:d6:f9:fd:8f:
    a0:7c:40:33:4c:90:1e:43:b2:14:fa:d3:82:00:b0:
    88:df:71:43:29:f7:7b:de:b4:87:0a:52:80:43:f7:
    1a:c9:1f:0f:c1:7c:27:00:45:5b:c4:a3:c3:a0:4a:
    14:53:7b:f6:33:54:44:70:41:8c:50:70:a1:eb:50:
    7a:52:87:6d:6c:84:2e:e5:48:86:9c:60:48:96:8e:
    48:2c:4b:13:87:af:94:cf:56:c1:74:e9:f5:4b:f9:
    16:32:fe:3a:65:a1:ef:eb:99:97:1a:10:e6:7f:6e:
    f3:18:1f:1f:67:f7:2a:38:0b:4b:00:70:0b:34:37:
    8c:04:56:02:ec:41:43:35:0c:c5:0c:8a:b2:91:00:
    7c:39:95:65:1e:b4:49:44:f6:af:87:d9:27:1e:a2:
    74:b3:5d:98:ac:da:53:fe:f6:9b:ee:15:fa:34:29:
    14:48:90:d6:61:31:43:3e:c0:30:f6:59:bb:b8:00:
    3b:98:a3:e8:d5:73:f1:ff:f5:23:d5:ac:87:57:ce:
    18:9f:35:1f:c2:4d:ba:44:05:b4:e3:ba:47:17:6a:
    76:5c:84:5c:f9:ce:83:19:87:ff:67:5b:82:24:4a:
    35:99:b1:91:7c:43:c9:84:2f:d4:1d:cf:6f:23:84:
    13:ae:59:28:66:a5:da:a4:5d:14:a4:04:69:21:4b:
    dd:d3:68:a3:16:0e:5f:23:4d:51:12:10:e3:f5:24:
    38:51:67:bf:cf:73:10:9a:b3:44:ff:87:bf:23:47:
    db:36:f0:c4:4a:6a:16:21:1f:ec:3c:1e:a9:c8:84:
    b4:2e:e5:d7:a1:d7:29:57:69:be:67:e9:1f:f5:66:
    dc:4b:c8:89:58:53:cd:c3:63:5e:58:86:48:db:71:
    3b:9f:25:1a:91:27:93:bf:1f:20:49:fc:b9:5b:ea:
    ba:b5:29:28:f2:a2:10:c5:ed:1c:fa:75:11:d2:22:
    7a:fd:50:be:56:e7:13:b1:a6:59:3b:aa:4b:8e:54:
    4b:1a:10:d8:6b:9c:46:ce:d8:7e:9a:f5:e7:28:62:
    6b:25:7c:ad:e6:64:4c:c0:4e:dc:1a:d8:c6:20:68:
    8b:3a:7a:8d:86:df:2e:e5:ab:39:7d:a1:3a:84:19:
    55:9b:46:2c:81:19:77:2c:2f:ca:6f:49:e0:92:98:
    c5:36:5a:db:4c:d8:58:04:4c:af:17:38:0e:2c:b1:
    21:1d:8b:88:69:69:fa:de:e7:fc:f0:9a:1a:71:1b:
    5e:68:51:b1:ef:44:1c:d6:a4:2d:55:93:b8:4c:e7:
    e2:dc:5b:99:ed:79:3a:02:6e:4e:61:32:62:03:a8:
    be:b2:4d
Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Basic Constraints:
  CA:FALSE
Netscape Comment:
  OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
  EC:FC:DC:5C:A5:50:1B:AE:B5:04:40:E3:C6:72:63:9F:F8:34:A3:89
X509v3 Authority Key Identifier:
  keyid:3B:36:FF:19:37:DF:12:D1:78:CC:F7:88:13:25:3D:CD:62:59:0F:9B
Signature Algorithm: sha256WithRSAEncryption
  6d:f0:74:43:a7:9a:1b:99:ee:9e:51:e8:9c:5f:4c:f3:c3:f4:
  ff:f5:5c:b4:e2:33:07:ed:fd:d5:09:e8:6f:b5:83:51:66:58:
  53:66:04:75:46:6b:5b:2a:ac:d8:54:78:df:de:d0:fc:6c:57:
  6e:0c:c9:3c:cf:62:64:63:cd:f3:a2:5b:fe:57:8c:12:7a:08:
  6a:53:b0:de:0e:c2:c7:35:9c:78:d5:27:98:05:b3:8c:54:fa:
  4f:0a:34:fc:4d:5f:14:93:5a:b1:86:2d:4a:74:e4:d8:83:e7:
  99:dd:36:56:a0:11:ff:d8:68:d2:62:13:72:96:4f:66:08:7c:
  f7:ce:7f:8f:2a:7d:c0:fc:fe:bf:1a:38:43:cc:01:aa:46:e4:
  ac:d7:be:65:51:a6:bc:6c:80:ed:54:76:0b:b8:b8:4b:a1:85:
  d5:9b:14:2b:65:19:cf:0f:65:5f:32:b8:9e:36:5c:c2:6f:92:
  7c:c5:db:90:11:40:db:e7:37:23:c5:78:1f:c0:09:c5:11:b8:
  06:30:3b:95:82:17:3e:49:51:7e:91:45:2e:a6:d1:89:1f:49:
  dc:6d:a9:1d:92:53:bd:ff:a2:cf:7b:8e:40:ab:96:4a:82:fd:
  5e:c7:04:53:2d:55:0c:78:57:03:5c:c1:6c:b1:12:ce:5f:be:
  10:9f:76:2c:5a:79:5a:a3:3c:68:ee:8c:69:c8:ac:cc:c5:09:
  fb:19:50:cb:5e:61:8a:9b:47:74:60:68:2a:94:49:6f:0f:22:
  78:6d:9b:ff:76:24:60:d1:ee:c6:59:59:e2:6f:f2:69:3f:ae:
  43:53:54:a1:ab:e0:ae:30:13:82:64:9c:ce:c1:fa:de:6a:7d:
  bb:94:a4:05:6e:03:84:d2:f1:f2:5d:1d:45:4c:bb:88:0b:6a:
  b6:74:8b:4b:fb:cf:99:6e:1d:ed:df:67:97:92:a7:0d:08:a7:
  57:be:7e:6e:06:13:f7:42:12:7a:05:aa:a9:1c:1b:1d:86:73:
  00:66:a7:07:88:d0:2d:eb:2b:dc:6e:e0:61:15:d9:ff:43:65:
  2d:99:c1:e6:80:ea:26:c4:08:ae:3c:12:ef:f0:6e:15:00:33:
  53:6a:c1:e3:14:5c:f3:ec:df:72:c1:ee:ca:ff:f6:c9:51:22:
  79:62:af:00:07:92:7c:0c:75:17:98:1d:b2:43:b2:fa:a7:41:
  d4:64:8e:3e:47:1c:f6:a0:aa:e3:30:d1:d7:5c:91:5c:ea:80:
  dc:31:8f:a2:40:fc:ac:4d:05:02:cb:c3:b4:f6:b5:98:58:7e:
```

```
31:c2:0c:85:1b:a4:95:ed:77:bb:dc:95:12:81:45:6c:5c:2f:
c4:7e:d3:86:9c:b0:1a:d4
```

If a component (director, file daemon, storage daemon or console) has **TLS Allowed CN = arrakis.example.com**, it will accept only connections from peers that have "CN=arrakis.example.com" in the subject field of their certificate. The **TLS Allowed CN** directive can have a list of values like **TLS Allowed CN = darkstar.example.com, arrakis.example.com**. This will permit the clients with CN Subject field equal to darkstar.example.com or arrakis.example.com to be allowed for connection. The below examples are configured with **TLS Allowed CN** to show where and how to configure this directive.

44.4.1 Enable TLS Communications Encryption Between Console and Director

Director and Console are on the Same "darkstar" Host

1. If you are using an anonymous console:
   You only need to define the TLS directives in the Director resource of both bacula-dir.conf and bconsole.conf files.
   - In bacula-dir.conf:

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = darkstar.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

   - In bconsole.conf:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

2. If a named console is used:
   You only need to define the TLS directives in the Console resource of both bacula-dir.conf and bconsole.conf.
   - In bacula-dir.conf:
     There is no need to configure TLS in the Director resource as for option 1

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
```

```
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
}
```

Instead, the named Console resource has the TLS configuration:

```
Console {
  Name = darkstar-con
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bconsole.conf:
  There is no need to configure TLS in the Director resource as for option (1) on the previous page:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
}
```

Instead, the Console resource contains the TLS configuration:

```
Console {
  Name = darkstar-con
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Director and Console are on Different Hosts

If you have your Bacula **Console** installed on another host than the Director one, then it is more likely that different public certificate and private key files for director and for console are used. Let's consider "darkstart-dir" director on "darkstar.example.com" and "arrakis-con" console on "arrakis.example.com".

44.4.2 Enable TLS Communications Encryption Between Console and Director

(1) If you're using an anonymous console:

- In bacula-dir.conf:

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
```

```
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = darkstar.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bconsole.conf`:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

2. If you are using a named console:

- In `bacula-dir.conf`:
  There is no need to configure TLS in the `Director` resource as for option 1 on the facing page:

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
}
```

  Instead, the `Console` resource has the TLS configurations:

```
Console {
  Name = arrakis-con
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Allowed CN = arrakis.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bconsole.conf`: It is not needed to configure TLS in the `Director` resource as for 1 on the preceding page:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
}
```

  Instead, the `Console` resource needs the TLS configuration:

```
Console {
  Name = arrakis-con
```

```
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

44.4.3 Enable TLS Communications Encryption between Director and File Daemon

Director and File Daemon are on the Same "darkstar" Host

- In bacula-dir.conf:

```
Client {
  Name = darkstar-fd
  Address = darkstar.example.com
  FD Port = 9112
  Catalog = MyCatalog
  Password = "password"
  AutoPrune = no
  Maximum Concurrent Jobs = 4
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-fd.conf:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = darkstart.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Director and File Daemon are on Different Hosts

Let's consider "darkstart-dir" director at "darkstar.example.com" and the "arrakis-fd" file daemon on "arrakis.example.com".

- In bacula-dir.conf:

```
Client {
  Name = arrakis-fd
  Address = arrakis.example.com
  FD Port = 9112
  Catalog = MyCatalog
  Password = "password"
  AutoPrune = no
  Maximum Concurrent Jobs = 4
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-fd.conf:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = arrakis.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

44.4.4 Enable TLS Communications Encryption Between Director and Storage Daemon

Director and Storage Daemon are on the Same "darkstar" Host

- In bacula-dir.conf:

```
Storage {
  Name = VTL-storage
  SD Port = 9113
  Address = darkstar.example.com
  Password = "password"
  Device = "Virtual Tape Library"
  Autochanger = yes
  Media Type = VTL
  Maximum Concurrent Jobs = 30
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = darkstart.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Director and Storage Daemon are on Different Hosts

Let's consider the "darkstart-dir" director on "darkstar.example.com" and the "caladan-sd" storage daemon running on "caladan.example.com".

- In bacula-dir.conf:

```
Storage {
  Name = VTL-storage
  SD Port = 9113
  Address = caladan.example.com
```

```
  Password = "password"
  Device = "Virtual Tape Library"
  Autochanger = yes
  Media Type = VTL
  Maximum Concurrent Jobs = 30
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = caladan.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

44.4.5 Enable TLS Communications Encryption Between File Daemon and Storage Daemon

Let's consider "darkstar-fd" (on "darkstar.example.com") and "arrakis-fd" (at "arrakis.example.com") clients, needing to connect to "caladan-sd" storage daemon at "caladan.example.com" using TLS.

- In `bacula-fd.conf` file at darkstar.example.com:

```
FileDaemon {
  Name = darkstar-fd
  FD Port = 9112
  FD Address = darkstar.example.com
  WorkingDirectory = /usr/local/bacula/working
  Pid Directory = /var/run
  Maximum Concurrent Jobs = 10
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bacula-fd.conf` file at "arrakis.example.com":

```
FileDaemon {
  Name = arrakis-fd
  FD Port = 9112
  FD Address = arrakis.example.com
  WorkingDirectory = /usr/local/bacula/working
  Pid Directory = /var/run
  Maximum Concurrent Jobs = 10
  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

- In bacula-sd.conf file at "caladan.example.com":

```
Storage {
  Name = caladan-sd
  SD Port = 9113
  SD Address = caladan.example.com
  WorkingDirectory = "/usr/local/bacula/working"
  Pid Directory = "/var/run"
  Maximum Concurrent Jobs = 40
  TLS Enable = yes
  TLS Require = yes
  TLS Allowed CN = darkstar.example.com , arrakis.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Please note that the **TLS Allowed CN** directive is only configured on the Storage Daemon side. This is due to the fact that the communication in this case is always between the Storage Daemon acting as a "TLS server" and the File Daemon acting as a "TLS client". In this case, the client is the peer in the TLS communications context. The CN in the client's certificate subject will be checked by the Storage Daemon if it is an "*Allowed CN*". in the case of the above example, **TLS Allowed CN = darkstar.example.com, arrakis.example.com** is allowing "darkstar-fd" (at darkstar.example.com) and "arrakis-fd" (at arrakis.example.com) clients to connect to "caladan-sd" storage daemon at "caladan.example.com" using TLS.

44.4.6 Using Certificates Issued by Different Root CAs

It is possible to have a TLS environment that uses certificates issued by different CAs. In the above examples, we have been using only one root CA:

```
# openssl genrsa -out ./keys/root_key.pem 4096
# openssl req -new -x509 -batch -config ./openssl.cnf -key ./keys/root_key.pem -days 36500 -out ./certs/root_c
```

In the case of the use of multiple root CA in your Bacula environment, there are two possible ways to configure:

1. to concatenate all the root CA certificates into one `.pem` file used in the **TLS CA Certificate File** directive:

```
# cat root_cert_ca1.pem root_cert_ca2.pem root_cert_ca3.pem > root_cert_ca.pem
```

   - use **TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert_ca.pem** in resources TLS directives definitions.

2. use the **TLS CA Certificate Dir** directive instead. In this case, the certificates should have OpenSSL-compatible hashes. Please find below an example:

```
root@darkstar:/opt/bacula/ssl/certs# ls -l
...
lrwxrwxrwx 1 root root 19 Oct 25 23:10 7293a8c5.0 -> root_ca1_cert.pem
lrwxrwxrwx 1 root root 19 Oct 25 23:10 8fb0c2b0.0 -> root_ca2_cert.pem
lrwxrwxrwx 1 root root 18 Oct 25 23:10 a6476ecf.0 -> root_ca3_cert.pem
-rw-r--r-- 1 root root 2134 Sep 7 16:03 root_ca1_cert.pem
-rw-r--r-- 1 root root 2134 Sep 7 15:56 root_ca2_cert.pem
-rw-r--r-- 1 root root 2134 Sep 7 11:47 root_ca3_cert.pem
```

   - **TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert_ca.pem** in resources TLS directives definitions.

44.4.7 Using TLS Authenticate to Enable a TLS Authentication Between Daemons

Bacula by default uses CRAM-**MD5!** authentication using the **Password** directive as a shared secret for each daemon. In addition to this, you can also use TLS to provide a more secure authentication between the daemons. This is achieved by using the **TLS Authenticate = yes** directive in the main daemon configuration resource in addition to configuring **TLS Enable**, **TLS Require**, **TLS Certificate CA File**, **TLS Certificate** and **TLS Key** directives. Please find examples below.

Please notice a very important feature of enabling **TLS Authenticate** to your daemons: **if you enable the TLS authentication, the TLS encryption will be turned off and communication between the daemons will be done without Encryption**.

Enabling TLS Authentication Between Director and Console

1. If you're using an anonymous console:
   You only need to define the TLS directives in the `Director` resource of both `bacula-dir.conf` and `bconsole.conf` files.

   - In `bacula-dir.conf`:

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

   - In `bconsole.conf`:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

2. If you are using a named console:
   You only need to define the TLS directives in the `Console` resource of both `bacula-dir.conf` and `bconsole.conf`.

   - In `bacula-dir.conf`:
     There is no need to configure TLS in the `Director` resource as for option 1:

```
Director {
  Name = darkstar-dir
  DIR Port = 9111
  DIR Address = darkstar.example.com
  QueryFile = "/usr/local/bacula/scripts/query.sql"
  WorkingDirectory = "/usr/local/bacula/working"
```

```
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 10
  Password = "password"
  Messages = Daemon
}
```

Instead, the `Console` resource has the TLS configurations:

```
Console {
  Name = darkstar-con
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bconsole.conf`:
  There is no need to configure TLS in the `Director` resource as for option (1) on the preceding page:

```
Director {
  Name = darkstar-dir
  DIRport = 9111
  Address = darkstar.example.com
  Password = "password"
}
```

Instead, the `Console` resource has the TLS configurations:

```
Console {
  Name = darkstar-con
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Enabling TLS Authentication Between Director and Client

Let's consider "darkstart-dir" director at "darkstar.example.com and" and the "arrakis-fd" file daemon at "arrakis.example.com".

- In `bacula-dir.conf`:

```
Client {
  Name = arrakis-fd
  Address = arrakis.example.com
  FD Port = 9112
  Catalog = MyCatalog
  Password = "password"
  AutoPrune = no
  Maximum Concurrent Jobs = 4
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bacula-fd.conf`:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Authenticate = yes
  TLS Allowed CN = arrakis.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

Enabling TLS Authentication Between Director and Storage

Let's consider "darkstart-dir" director at "darkstar.example.com" and the "caladan-sd" storage daemon on "caladan.example.com".

- In `bacula-dir.conf`:

```
Storage {
  Name = VTL-storage
  SD Port = 9113Address = caladan.example.com
  Password = "password"
  Device = "Virtual Tape Library"
  Autochanger = yes
  Media Type = VTL
  Maximum Concurrent Jobs = 30
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
  Name = darkstar-dir
  Password = "password"
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Authenticate = yes
  TLS Allowed CN = caladan.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Enabling TLS Authentication Between Client and Storage

Let's consider "darkstar-fd" (on "darkstar.example.com") and "arrakis-fd" (at "arrakis.example.com") clients need to connect to the "aladan-sd" storage daemon running on "caladan.example.com" using TLS.

- In `bacula-fd.conf` file at darkstar.example.com:

```
FileDaemon {
  Name = darkstar-fd
  FD Port = 9112
  FD Address = darkstar.example.com
  WorkingDirectory = /usr/local/bacula/working
```

```
  Pid Directory = /var/run
  Maximum Concurrent Jobs = 10
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In `bacula-fd.conf` file at "arrakis.example.com":

```
FileDaemon {
  Name = arrakis-fd
  FD Port = 9112
  FD Address = arrakis.example.com
  WorkingDirectory = /usr/local/bacula/working
  Pid Directory = /var/run
  Maximum Concurrent Jobs = 10
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

- In bacula-sd.conf file at "caladan.example.com":

```
Storage {
  Name = caladan-sd
  SD Port = 9113
  SD Address = caladan.example.com
  WorkingDirectory = "/usr/local/bacula/working"
  Pid Directory = "/var/run"
  Maximum Concurrent Jobs = 40
  TLS Enable = yes
  TLS Require = yes
  TLS Authenticate = yes
  TLS Allowed CN = darkstar.example.com , arrakis.example.com
  TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
  TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_cert.pem
  TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Chapter 45

Data Encryption

Bacula permits file data encryption and signing within the File Daemon (or Client) prior to sending data to the Storage Daemon. Upon restoration, file signatures are validated and any mismatches are reported. At no time does the Director or the Storage Daemon have access to unencrypted file contents.

It is very important to specify what this implementation does NOT do:

- There is one important restore problem to be aware of, namely, it's possible for the director to restore new keys or a Bacula configuration file to the client, and thus force later backups to be made with a compromised key and/or with no encryption at all. You can avoid this by not changing the location of the keys in your Bacula File daemon configuration file, and not changing your File daemon keys. If you do change either one, you must ensure that no restore is done that restores the old configuration or the old keys. In general, the worst effect of this will be that you can no longer connect the File daemon.
- The implementation does not encrypt file metadata such as file path names, permissions, and ownership. Extended attributes are also currently not encrypted. However, Mac OSX resource forks are encrypted.

Encryption and signing are implemented using RSA private keys coupled with self-signed X.509 public certificates. This is also sometimes known as Public-Key Infrastructure (PKI).

Each File Daemon should be given its own unique private/public key pair. In addition to this key pair, any number of "Master Keys" may be specified – these are key pairs that may be used to decrypt any backups should the File Daemon key be lost. Only the Master Key's public certificate should be made available to the File Daemon. Under no circumstances should the Master Private Key be shared or stored on the Client machine.

The Master Keys should be backed up to a secure location, such as a CD placed in a in a fire-proof safe or bank safety deposit box. The Master Keys should never be kept on the same machine as the Storage Daemon or Director if you are worried about an unauthorized party compromising either machine and accessing your encrypted backups.

While less critical than the Master Keys, File Daemon Keys are also a prime candidate for off-site backups; burn the key pair to a CD and send the CD home with the owner of the machine.

NOTE!!! If you lose your encryption keys, backups will be unrecoverable. **ALWAYS** store a copy of your master keys in a secure, off-site location.

The basic algorithm used for each backup session (Job) is:

1. The File daemon generates a session key.

2. The FD encrypts that session key via PKE for all recipients (the file daemon, any master keys).
3. The FD uses that session key to perform symmetric encryption on the data.

45.1 Building Bacula with Encryption Support

To build Bacula with encryption support, you will need the OpenSSL libraries and headers installed. When configuring Bacula, use:

```
./configure --with-openssl ...
```

Please note, the Bacula Enterprise binaries are all built with encryption enabled.

45.2 Encryption Technical Details

The implementation uses 128bit AES-CBC, with RSA encrypted symmetric session keys. The RSA key is user supplied. If you are running OpenSSL 0.9.8 or later, the signed file hash uses SHA-256 – otherwise, SHA-1 is used.

End-user configuration settings for the algorithms are not currently exposed – only the algorithms listed above are used. However, the data written to Volume supports arbitrary symmetric, asymmetric, and digest algorithms for future extensibility, and the back-end implementation currently supports:

- Symmetric Encryption:
  - 128, 192, and 256-bit AES-CBC
  - Blowfish-CBC
- Asymmetric Encryption (used to encrypt symmetric session keys):
  - RSA
- Digest Algorithms:
  - MD5
  - SHA1
  - SHA256
  - SHA512

The various algorithms are exposed via an entirely re-usable, OpenSSL-agnostic API (ie, it is possible to drop in a new encryption backend). The Volume format is DER-encoded ASN.1, modeled after the Cryptographic Message Syntax from RFC 3852. Unfortunately, using CMS directly was not possible, as at the time of coding a free software streaming DER decoder/encoder was not available.

45.3 Concepts

Data encription is configured in Bacula only in the File Daemon. The Job report that is produced at the end of each job has a line indicating whether or not encryption was enabled:

```
VSS:         no
Encryption:  no/yes    <===
Accurate:    no
...
```

45.4 Generating Private/Public Encryption Keys

Generate a Master Key Pair (once) with:

```
openssl genrsa -out master.key 2048
openssl req -new -key master.key -x509 -out master.cert
```

Save these files (`master.key` and `master.cert`) as you will need them to create File Daemon keys or in case you loose the File Daemon keys.

Generate a File Daemon Key Pair for each FD:

```
openssl genrsa -out fd-example.key 2048
openssl req -new -key fd-example.key -x509 -out fd-example.cert
cat fd-example.key fd-example.cert >fd-example.pem
```

When configuring the File Daemon (see section 23 on page 301), you will need the keys you just generated here, so you will need to transmit them to the machine where the FD is installed.

Note, there seems to be a lot of confusion around the file extensions given to these keys. For example, a `.pem` file can contain all the following: private keys (RSA and DSA), public keys (RSA and DSA) and (X.509) certificates. It is the default format for OpenSSL. It stores data Base64 encoded DER format, surrounded by ASCII headers, so is suitable for text mode transfers between systems. A `.pem` file may contain any number of keys either public or private. We use it in cases where there is both a public and a private key.

Typically, above we have used the `.cert` extension to refer to X.509 certificate encoding that contains only a single public key.

45.5 Example Data Encryption Configuration

When configuring the FD, use the keys generated above in a FD configuration file that will look something like the following:

`bacula-fd.conf`

```
FileDaemon {
   Name = example-fd
   FDport = 9102                  # where we listen for the director
   WorkingDirectory = /opt/bacula/working
   Pid Directory = /var/run
   Maximum Concurrent Jobs = 20

   PKI Signatures = Yes            # Enable Data Signing
   PKI Encryption = Yes            # Enable Data Encryption
   PKI Keypair = "/opt/bacula/etc/fd-example.pem"    # Public and Private Keys
   PKI Master Key = "/opt/bacula/etc/master.cert"    # ONLY the Public Key
}
```

You must restart your File Daemon after making this change to the `bacula-fd.conf` file.

Note : the PKIMasterKey directive is not mandatory, but if used will allow you to decrypt the files if ever the FD PKIKeypair is lost. If you loose the FD's PKIKeypair, you will not be able to recover your data unless you have used a PKIMasterKey.

45.6 Decrypting with a Master Key

It is preferable to retain a secure, non-encrypted copy of the client's own encryption keypair. However, should you lose the client's keypair, recovery with the master keypair is possible.

First create a keypair with:

```
cat master.key master.cert >master.pem
```

Then modify your File Daemons configuration file to use the master keypair:

```
FileDaemon {
   Name = example-fd
   FDport = 9102                  # where we listen for the director
   WorkingDirectory = /opt/bacula/working
   Pid Directory = /var/run
   Maximum Concurrent Jobs = 20

   PKI Signatures = Yes            # Enable Data Signing
   PKI Encryption = Yes            # Enable Data Encryption
   PKI Keypair = "/opt/bacula/etc/master.pem"   # Master Public and Private Keys
}
```

Restart your File Daemon and you should be able to recover your lost files.

Chapter 46

Using Bacula to Improve Computer Security

Since Bacula maintains a catalog of files, their attributes, and either SHA1 or MD5 signatures, it can be an ideal tool for improving computer security. This is done by making a snapshot of your system files with a **Verify** Job and then checking the current state of your system against the snapshot, on a regular basis (e.g. nightly).

The first step is to set up a **Verify** Job and to run it with:

```
Level = InitCatalog
```

The **InitCatalog** level tells **Bacula** simply to get the information on the specified files and to put it into the catalog. That is your database is initialized and no comparison is done. The **InitCatalog** is normally run one time manually.

Thereafter, you will run a Verify Job on a daily (or whatever) basis with:

```
Level = Catalog
```

The **Level = Catalog** level tells Bacula to compare the current state of the files on the Client to the last **InitCatalog** that is stored in the catalog and to report any differences. See the example below for the format of the output.

You decide what files you want to form your "snapshot" by specifying them in a **FileSet** resource, and normally, they will be system files that do not change, or that only certain features change.

Then you decide what attributes of each file you want compared by specifying comparison options on the **Include** statements that you use in the **FileSet** resource of your **Catalog** Jobs.

46.1 The Details

In the discussion that follows, we will make reference to the Verify Configuration Example that is included below in the **A Verify Configuration Example** section. You might want to look it over now to get an idea of what it does.

The main elements consist of adding a schedule, which will normally be run daily, or perhaps more often. This is provided by the **VerifyCycle** Schedule, which runs at 5:05 in the morning every day.

Then you must define a Job, much as is done below. We recommend that the Job name contain the name of your machine as well as the word **Verify** or **Check**. In our example, we named it **MatouVerify**. This will permit you to easily identify your job when running it from the Console.

You will notice that most records of the Job are quite standard, but that the **FileSet** resource contains **verify=pins1** option in addition to the standard **signature=SHA1** option. If you don't want SHA1 signature comparison, and we cannot imagine why not, you can drop the **signature=SHA1** and none will be computed nor stored in the catalog. Or alternatively, you can use **verify=pins5** and **signature=MD5**, which will use the MD5 hash algorithm. The MD5 hash computes faster than SHA1, but is cryptographically less secure.

The **verify=pins1** is ignored during the **InitCatalog** Job, but is used during the subsequent **Catalog** Jobs to specify what attributes of the files should be compared to those found in the catalog. **pins1** is a reasonable set to begin with, but you may want to look at the details of these and other options. They can be found in the FileSet Resource section of this manual. Briefly, however, the **p** of the **pins1** tells Verify to compare the permissions bits, the **i** is to compare inodes, the **n** causes comparison of the number of links, the **s** compares the file size, and the **1** compares the SHA1 checksums (this requires the **signature=SHA1** option to have been set also).

You must also specify the **Client** and the **Catalog** resources for your Verify job, but you probably already have them created for your client and do not need to recreate them, they are included in the example below for completeness.

As mentioned above, you will need to have a **FileSet** resource for the Verify job, which will have the additional **verify=pins1** option. You will want to take some care in defining the list of files to be included in your **FileSet**. Basically, you will want to include all system (or other) files that should not change on your system. If you select files, such as log files or mail files, which are constantly changing, your automatic Verify job will be constantly finding differences. The objective in forming the FileSet is to choose all unchanging important system files. Then if any of those files has changed, you will be notified, and you can determine if it changed because you loaded a new package, or because someone has broken into your computer and modified your files. The example below shows a list of files that I use on my Red Hat 7.3 system. Since I didn't spend a lot of time working on it, it probably is missing a few important files (if you find one, please send it to me). On the other hand, as long as I don't load any new packages, none of these files change during normal operation of the system.

46.2 Running the Verify

The first thing you will want to do is to run an **InitCatalog** level Verify Job. This will initialize the catalog to contain the file information that will later be used as a basis for comparisons with the actual file system, thus allowing you to detect any changes (and possible intrusions into your system).

The easiest way to run the **InitCatalog** is manually with the console program by simply entering `run`. You will be presented with a list of Jobs that can be run, and you will choose the one that corresponds to your Verify Job, **MatouVerify** in this example.

```
The defined Job resources are:
     1: MatouVerify
     2: kernsrestore
     3: Filetest
     4: kernsave
Select Job resource (1-4): 1
```

Next, the console program will show you the basic parameters of the Job and ask you:

```
Run Verify job
JobName:  MatouVerify
FileSet:  Verify Set
Level:    Catalog
Client:   MatouVerify
Storage:  DLTDrive
Verify Job:
Verify List: /tmp/regress/working/MatouVerify.bsr
OK to run? (yes/mod/no): mod
```

Here, you want to respond **mod** to modify the parameters because the Level is by default set to **Catalog** and we want to run an **InitCatalog** Job. After responding **mod**, the console will ask:

```
Parameters to modify:
     1: Level
     2: Storage
     3: Job
     4: FileSet
     5: Client
     6: When
     7: Priority
     8: Pool
     9: Verify Job
Select parameter to modify (1-5): 1
```

you should select number 2 to modify the **Level**, and it will display:

```
Levels:
     1: Initialize Catalog
     2: Verify Catalog
     3: Verify Volume to Catalog
     4: Verify Disk to Catalog
     5: Verify Volume Data
Select level (1-4): 1
```

Choose item 1, and you will see the final display:

```
Run Verify job
JobName:  MatouVerify
FileSet:  Verify Set
Level:    Initcatalog
Client:   MatouVerify
Storage:  DLTDrive
Verify Job:
Verify List: /tmp/regress/working/MatouVerify.bsr
OK to run? (yes/mod/no): yes
```

at which point you respond **yes**, and the Job will begin.

Thereafter the Job will automatically start according to the schedule you have defined. If you wish to immediately verify it, you can simply run a Verify **Catalog** which will be the default. No differences should be found.

To use a previous job, you can add `jobid=xxx` option in `run` command line. It will run the Verify job against the specified job.

```
*run jobid=1 job=MatouVerify
Run Verify job
JobName:     MatouVerify
Level:       Catalog
Client:      127.0.0.1-fd
FileSet:     Full Set
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
```

```
Verify Job:  MatouVerify.2010-09-08_15.33.33_03
Verify List: /tmp/regress/working/MatouVerify.bsr
When:        2010-09-08 15:35:32
Priority:    10
OK to run? (yes/mod/no):
```

46.3 What To Do When Differences Are Found

If you have setup your messages correctly, you should be notified if there are any differences and exactly what they are. For example, below is the email received after doing an update of OpenSSH:

```
HeadMan: Start Verify JobId 83 Job=RufusVerify.2002-06-25.21:41:05
HeadMan: Verifying against Init JobId 70 run 2002-06-21 18:58:51
HeadMan: File: /etc/pam.d/sshd
HeadMan:       st_ino   differ. Cat: 4674b File: 46765
HeadMan: File: /etc/rc.d/init.d/sshd
HeadMan:       st_ino   differ. Cat: 56230 File: 56231
HeadMan: File: /etc/ssh/ssh_config
HeadMan:       st_ino   differ. Cat: 81317 File: 8131b
HeadMan:       st_size  differ. Cat: 1202 File: 1297
HeadMan:       SHA1 differs.
HeadMan: File: /etc/ssh/sshd_config
HeadMan:       st_ino   differ. Cat: 81398 File: 81325
HeadMan:       st_size  differ. Cat: 1182 File: 1579
HeadMan:       SHA1 differs.
HeadMan: File: /etc/ssh/ssh_config.rpmnew
HeadMan:       st_ino   differ. Cat: 812dd File: 812b3
HeadMan:       st_size  differ. Cat: 1167 File: 1114
HeadMan:       SHA1 differs.
HeadMan: File: /etc/ssh/sshd_config.rpmnew
HeadMan:       st_ino   differ. Cat: 81397 File: 812dd
HeadMan:       st_size  differ. Cat: 2528 File: 2407
HeadMan:       SHA1 differs.
HeadMan: File: /etc/ssh/moduli
HeadMan:       st_ino   differ. Cat: 812b3 File: 812ab
HeadMan: File: /usr/bin/scp
HeadMan:       st_ino   differ. Cat: 5e07e File: 5e343
HeadMan:       st_size  differ. Cat: 26728 File: 26952
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/ssh-keygen
HeadMan:       st_ino   differ. Cat: 5df1d File: 5e07e
HeadMan:       st_size  differ. Cat: 80488 File: 84648
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/sftp
HeadMan:       st_ino   differ. Cat: 5e2e8 File: 5df1d
HeadMan:       st_size  differ. Cat: 46952 File: 46984
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/slogin
HeadMan:       st_ino   differ. Cat: 5e359 File: 5e2e8
HeadMan: File: /usr/bin/ssh
HeadMan:       st_mode  differ. Cat: 89ed File: 81ed
HeadMan:       st_ino   differ. Cat: 5e35a File: 5e359
HeadMan:       st_size  differ. Cat: 219932 File: 234440
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/ssh-add
HeadMan:       st_ino   differ. Cat: 5e35b File: 5e35a
HeadMan:       st_size  differ. Cat: 76328 File: 81448
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/ssh-agent
HeadMan:       st_ino   differ. Cat: 5e35c File: 5e35b
HeadMan:       st_size  differ. Cat: 43208 File: 47368
HeadMan:       SHA1 differs.
HeadMan: File: /usr/bin/ssh-keyscan
HeadMan:       st_ino   differ. Cat: 5e35d File: 5e96a
HeadMan:       st_size  differ. Cat: 139272 File: 151560
HeadMan:       SHA1 differs.
HeadMan: 25-Jun-2002 21:41
JobId:                 83
```

```
Job:                    RufusVerify.2002-06-25.21:41:05
FileSet:                Verify Set
Verify Level:           Catalog
Client:                 RufusVerify
Start time:             25-Jun-2002 21:41
End time:               25-Jun-2002 21:41
Files Examined:         4,258
Termination:            Verify Differences
```

At this point, it was obvious that these files were modified during installation of the RPMs. If you want to be super safe, you should run a **Verify Level=Catalog** immediately before installing new software to verify that there are no differences, then run a **Verify Level=InitCatalog** immediately after the installation.

To keep the above email from being sent every night when the Verify Job runs, we simply re-run the Verify Job setting the level to **InitCatalog** (as we did above in the very beginning). This will re-establish the current state of the system as your new basis for future comparisons. Take care that you don't do an **InitCatalog** after someone has placed a Trojan horse on your system!

If you have included in your **FileSet** a file that is changed by the normal operation of your system, you will get false matches, and you will need to modify the **FileSet** to exclude that file (or not to Include it), and then re-run the **InitCatalog**.

The FileSet that is shown below is what I use on my Red Hat 7.3 system. With a bit more thought, you can probably add quite a number of additional files that should be monitored.

46.4 A Verify Configuration Example

```
Schedule {
  Name = "VerifyCycle"
  Run = Level=Catalog sun-sat at 5:05
}
Job {
  Name = "MatouVerify"
  Type = Verify
  Level = Catalog                     # default level
  Client = MatouVerify
  FileSet = "Verify Set"
  Messages = Standard
  Storage = DLTDrive
  Pool = Default
  Schedule = "VerifyCycle"
}
#
# The list of files in this FileSet should be carefully
# chosen. This is a good starting point.
#
FileSet {
  Name = "Verify Set"
  Include {
    Options {
      verify=pins1
      signature=SHA1
    }
    File = /boot
    File = /bin
    File = /sbin
    File = /usr/bin
    File = /lib
    File = /root/.ssh
    File = /home/kern/.ssh
    File = /var/named
    File = /etc/sysconfig
    File = /etc/ssh
    File = /etc/security
    File = /etc/exports
    File = /etc/rc.d/init.d
```

```
    File = /etc/sendmail.cf
    File = /etc/sysctl.conf
    File = /etc/services
    File = /etc/xinetd.d
    File = /etc/hosts.allow
    File = /etc/hosts.deny
    File = /etc/hosts
    File = /etc/modules.conf
    File = /etc/named.conf
    File = /etc/pam.d
    File = /etc/resolv.conf
  }
  Exclude = { }
}
Client {
  Name = MatouVerify
  Address = lmatou
  Catalog = Bacula
  Password = ""
  File Retention = 80d                # 80 days
  Job Retention = 1y                  # one year
  AutoPrune = yes                     # Prune expired Jobs/Files
}
Catalog {
  Name = Bacula
  dbname = verify; user = bacula; password = ""
}
```

Chapter 47

Using Bacula Continous Data Protection

Continous Data Protection (CDP) also called continuous backup or real-time backup, refers to backup of Client data by automatically saving a copy of every change made to that data, essentially capturing every version of the data that the user saves. It allows the user or administrator to restore data to any point in time.

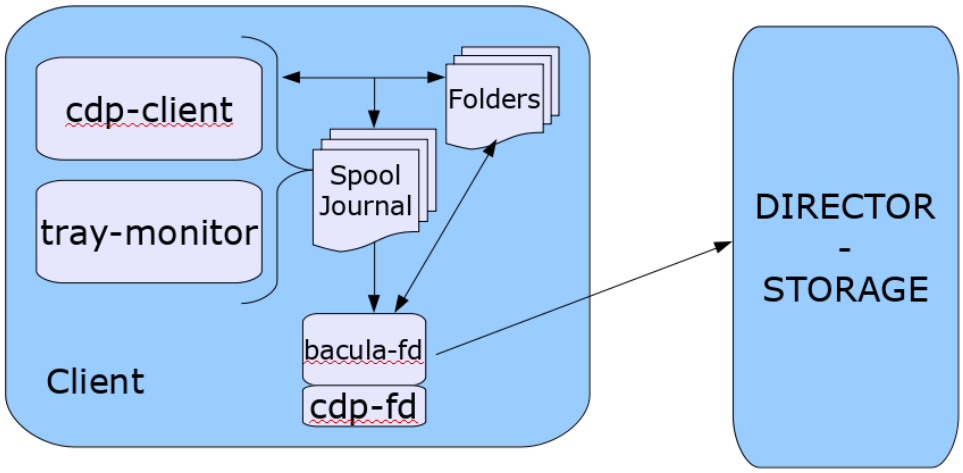


Figure 47.1: CDP Example

The Bacula Enterprise CDP feature is composed of two components. A application (`cdp-client` or `tray-monitor`) that will monitor a set of directories configured by the user and a Bacula FileDaemon plugin responsible to secure the data on a Bacula infrastructure.

The user application (`cdp-client` or `tray-monitor`) is responsible to monitor files and directories. When a modification is detected, a copy of the new data is done into a `spool` directory. At a regular interval, a Bacula backup job will contact the FileDaemon and will save all the files archived by the `cdp-client`. The local data can be restored at any time without a network connection to the Director.

47.1 Configuration

47.1.1 FileDaemon Configuration

As with all Bacula plugins, the **Plugin Directory** directive in the **FileDaemon** resource of the `bacula-fd.conf` file needs to be set:

```
FileDaemon {
  Name = test-fd
  ...
  Plugin Directory = /opt/bacula/plugins
}
```

47.1.2 Plugin Parameters

In order to configure the CDP Plugin, one of the following parameters must be set:

The following parameters are used in the CDP job.

- `user=<string>` specifies a user in the system. The CDP Plugin will guess the location of the journal file for this user. Not available on Windows.
- `group=<string>` specifies a group in the system. The CDP Plugin will guess the location of the journal files of the users from this group. Not available on Windows.
- `userHome=<string>` specifies the path to the cdp User Home. This is where the CDP Plugin will look for the journal file. On Windows, this is the User AppData/Roaming folder, for example: "C:/Users/Auser/AppData/Roaming".

47.1.3 Director Configuration

```
FileSet {
  Name = cdp
  Include {
    Options {
      compression = LZO
    }

    Plugin = "cdp: userHome=/home/user1"
  }
}

Job {
  Name = CDP
  Client = myclient-fd
  Type = Backup
  Pool = Default
  Schedule = Hourly      # Choose a schedule
  Messages = Standard
  FileSet = cdp
}
```

47.1.4 CDP Client Configuration

-f <dir> watches the directory <dir> for changes

-j <dir> sets journal directory to <dir>. Default value is <HOME>

-s <dir> sets spool directory to <dir>. Default value is <HOME>/.cdp-sdir

-d <nn> set debug level to <nn>

-h print help

47.1.5 Tray Monitor Configuration

You can setup the folders you wish to watch for changes by using the Tray Monitor.

Open the Tray Monitor options and click in the option Watch...:

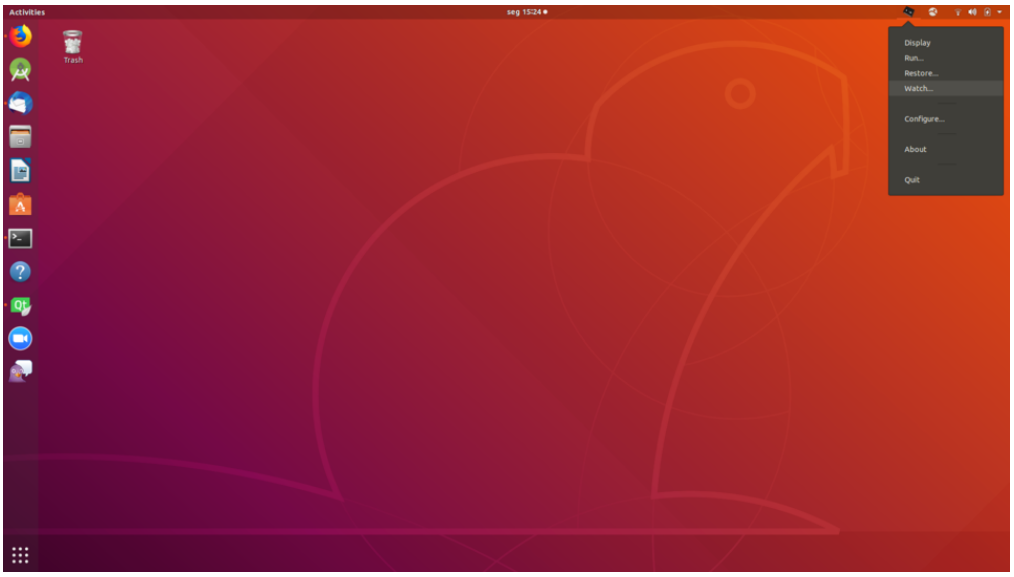


Figure 47.2: Open CDP Client

You should see a window displaying the watched folders. Click in the button Add:

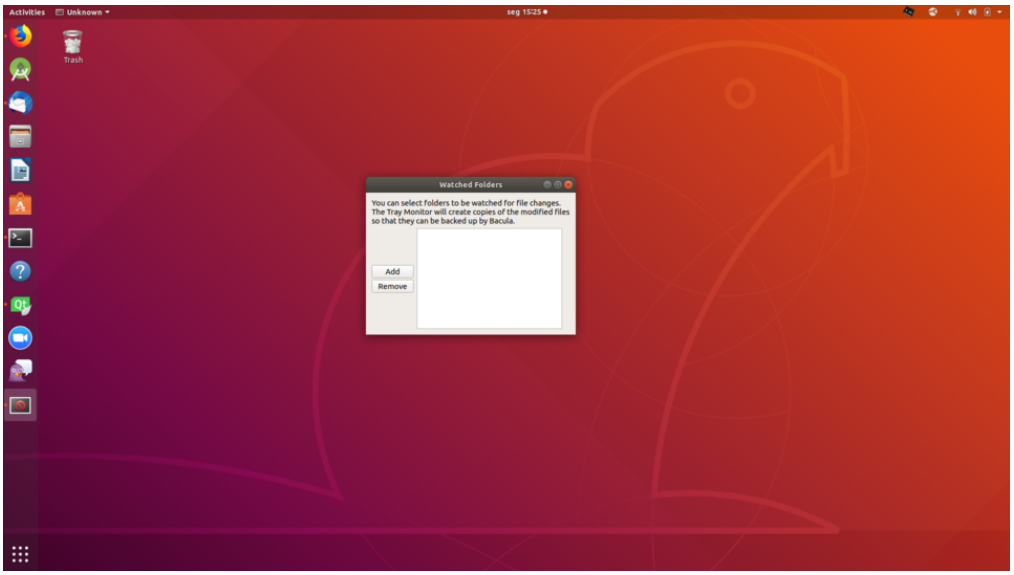


Figure 47.3: CDP Client Window

Select the folder you wish to watch:

You should see the watched folder listed, as in the image below:

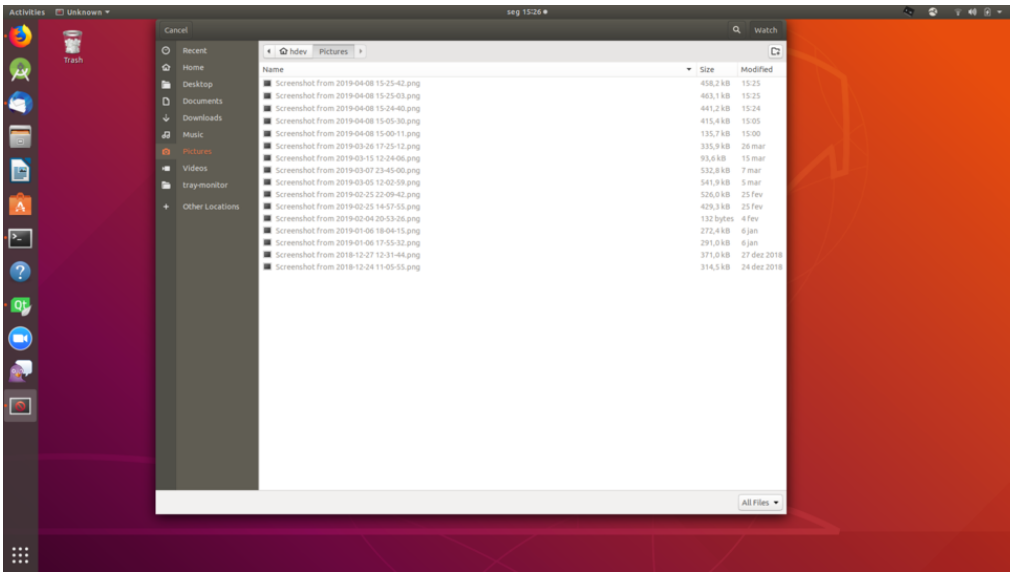


Figure 47.4: Watching a Folder

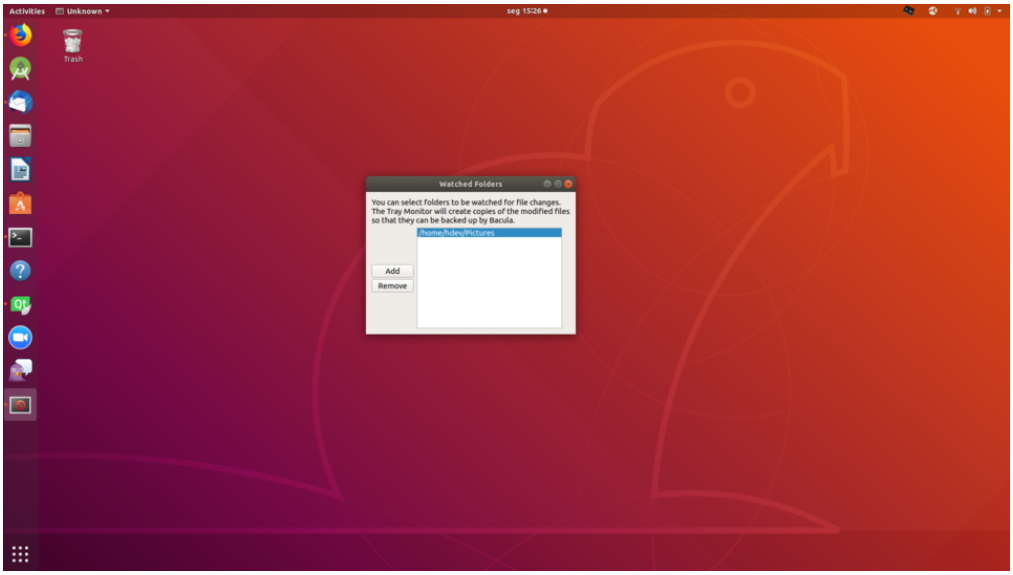


Figure 47.5: Result

47.1.6 The Spool Directory and Scheduled Backups

The Spool Directory contains all files from the watched folders that were created and / or changed. It keeps a copy of every version of the files.

An example of this directory can be seen below:

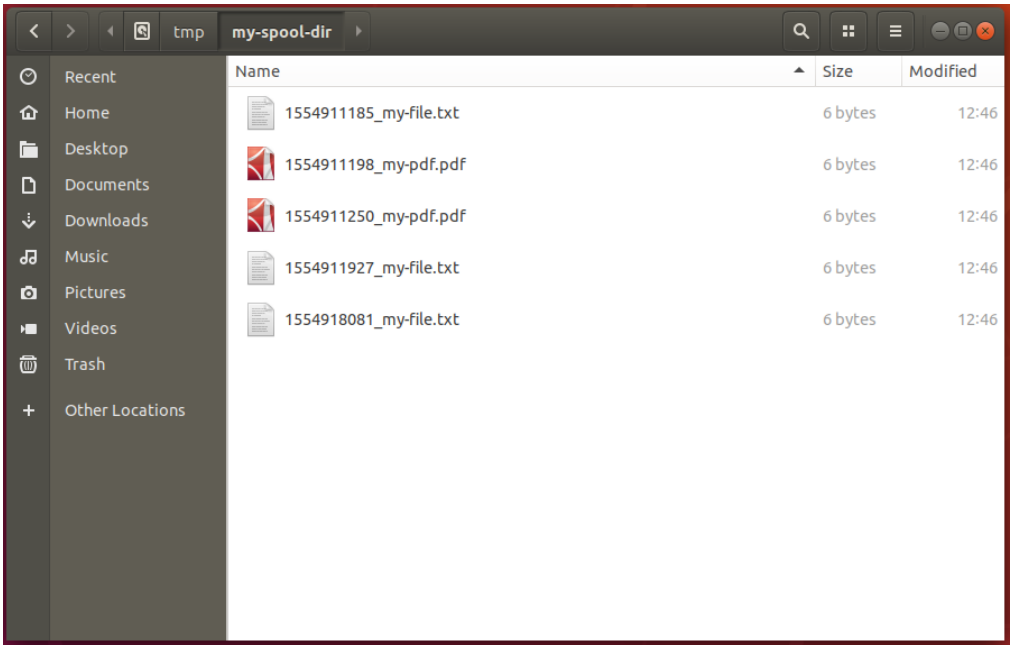


Figure 47.6: Result

You must schedule backup jobs with a proper Files in order to backup those files with Bacula. Following this example, the job configuration would be:

```
FileSet {
  Name = cdp
  Include {
    Options {
      compression = LZO
    }
    Plugin = "cdp: userHome=/home/user1"
  }
}

Job {
  Name = CDP
  Client = myclient-fd
  Type = Backup
  Pool = Default
  Schedule = Hourly      # Choose a schedule
  Messages = Standard
  FileSet = cdp
}
```

The Job reads the journal file to select which files from the Spool Directory should be backed up. After that, the files will be available to be restored at any time:

47.1.7 Important

It's necessary to keep either the `cdp-client` or the `tray-monitor` running in order to protect the directories being watched.

```
list files jobid=1
Using Catalog "MyCatalog"
+-------------------------------------------------------------------+
| filename                                                          |
+-------------------------------------------------------------------+
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-file.txt |
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-pdf.pdf  |
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-png.png  |
+-------------------------------------------------------------------+
```

Figure 47.7: Job Files

47.1.8 Limitations

- There is a limit in the number of folders that can be watched by default, which varies depending on your Operating System. You can increase that number by checking the correct procedure for your Operating System.
- On Linux OSes, the CDP client implementation relies on the inotify API, therefore being subject to it's limitations (for example, it doesn't work with NFS). On Windows, the client relies on the ReadDirectoryChangesW API.

Chapter 48

Plugins

48.1 Kubernetes Plugin

48.1.1 Features Summary

- Kubernetes cluster resources configuration backup
- Ability to restore single Kubernetes configuration resource
- Ability to restore Kubernetes resource configuration to local directory
- Kubernetes Persistent Volumes data backup and restore
- Ability to restore Kubernetes Persistent Volumes data to local directory
- Ability to use new Kubernetes CSI driver snapshot features to perform Persistent Volume data backup
- Ability to execute user defined commands on required Pod containers to prepare and clean data backup
- Configure Kubernetes workload backup requirements directly with Pod annotations

48.1.2 Glossary

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization".

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

Container is a runnable instance of an image and refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances.

Kubernetes Resource is a configuration entity which describe how a particular component should be executed and maintained. It includes core components like Pod, Service, Persistent Volume Claim or extended components like Replica Set, Deployment or Stateful Set.

K8S is short for Kubernetes and is used interchangeably.

Persistent Volume (PV) is a piece of storage in the cluster that has been dynamically provisioned or provisioned by an administrator who's life cycle is independent of any individual Pod that uses the PV.

Persistent Volume Claim (PVC) is a request for storage by a user which could be consumed by Pod. This resource associates a PV with a Pod.

Bacula Backup Proxy Pod is a Kubernetes service dedicated to PVC Data archive backup and restore operations which acts as a proxy application between Persistent Volume data and a Bacula File Daemon with the Kubernetes Plugin. This application requires a dedicated bacula-backup docker image to be deployed.

CSI is the Container Storage Interface. CSI is a standard for exposing arbitrary block and file storage systems to containerized workloads on Container Orchestration Systems (COs) like Kubernetes. Using CSI third-party storage providers can write and deploy plugins exposing new storage systems in Kubernetes without ever having to touch the core Kubernetes code.

48.1.3 Kubernetes Backup Overview

Containers are very light system level virtualization with less overhead because programs in virtual partitions use the operating system's normal system call interface and do not need to be subjected to emulation or run in an intermediate virtual machine. Kubernetes manages a set of containers to create a flexible execution environment for applications and services.

The Bacula Kubernetes plugin will save all the important Kubernetes resources which make up the application or service. This includes the following namespaced objects:

- Config Map
- Daemon Set
- Deployment
- Endpoint
- Limit Range
- Pod
- Persistent Volume Claim
- Pod Template
- Replica Set
- Replication Controller
- Resource Quota
- Secret
- Service
- Service Account
- Stateful Set
- PVC Data Archive[1]

and non namespaced objects:

- Namespace
- Persistent Volume
- Storage Class

[1]The PVC Data is not exact Kubernetes Object but represents archive of real data existed on selected PVC.

All namespaced objects which belong to the particular namespace are grouped together for easy backup data browsing and recovery.

Users and service accounts can be authorized to access the server API. This process goes through authentication, authorization and admission control. To be able to successfully backup the Kubernetes resources, it is required to have a user or service account with the correct permissions and rights to be successfully authenticated and authorized to access the API server and resources to be backed up.

For resource configuration backups, the user or service account must be able to read and list the resources. In the case of PVC Data backup, it is also required that the user or service account can create and delete pods because the plugin will need to create and delete the Bacula Backup Proxy Pod during the backup.

Please see the Kubernetes documentation for more details.

48.1.4 Kubernetes Persistent Volume Claim Backup

All Pods in Kubernetes are ephemeral and may be destroyed manually or by operations from controllers. Pods do not store data locally because stored data would be destroyed with a pod's life cycle management, so data is saved on Persistent Volumes using Persistent Volume Claim objects to control Volume Space availability.

This brings a new challenge to data backup. Fortunately most of the challenges found here are similar to standard bare metal or virtualized environments. As with bare metal and virtual machine environments, data stored in databases should be protected with dedicated Bacula plugins that take advantage of the database backup routines.

Please refer to the appropriate Bacula plugin whitepapers for more details on database backups.

On the other hand, most non-database applications store data as simple flat files we can backup as-is without forcing complicated transactions or data consistency procedures. This use case is handled directly with the Kubernetes plugin using a dedicated Bacula Backup Proxy Pod executed in the cluster.

If the container application is more complex, it is possible to execute commands inside the container to quiesce the application.

- before the snapshot
- after the snapshot
- after the backup of the container

A problem with command execution can abort the backup of the container with the `run.*.failonerror` annotation. You can find detailed description of this feature at 48.1.5 on the next page.

A Bacula Backup Proxy Pod is a service executed automatically by the Kubernetes plugin which manages secure access to Persistent Volume data for the plugin. It is executed on the Kubernetes cluster infrastructure and requires a network connection to the Kubernetes plugin for data exchange on backup and restore operations. No external cluster service like `NodePort`, `LoadBalancer`, `Ingress` or `Host Based Networking` configuration is required to use this feature.

It is also not required to permanently deploy and run this service on the cluster itself as it is executed on demand. The Bacula Backup Proxy Pod does not consume any valuable compute

resources outside of the backup window. You can even operate your Kubernetes backup solution (Bacula Edition service with Kubernetes plugin) directly from your on-premise backup infrastructure to backup a public Kubernetes cluster (it requires a simple port forwarding firewall rule) or use public backup infrastructure to backup on-premise Kubernetes cluster(s). Support for these varied architecture modes is built into the Kubernetes plugin. It is designed to be a *One-Click* solution for Kubernetes backups.

It is possible to backup and restore any PVC data including PVCs not attached to any running Kubernetes Pod.

48.1.5 CSI Snapshot Support

Kubernetes CSI Snapshot functionality support together with a bunch of other features was added. Starting from this version you can use CSI Snapshots to acquire a consistent data view of selected Volume. Additionally, you can configure remote command execution on a selected Container of the Pod. You can configure command execution just before or after a Pod backup and just after snapshot creation.

Our plugin uses the volume clone api when doing a volume snapshot. CSI drivers may or may not have implemented the volume cloning functionality.

48.1.6 Kubernetes Pod Annotations

The **CSI Snapshot Support** feature described in 48.1.5 comes with a configuration of Volume data backup using Kubernetes Pod annotations. This feature allows you to define what volumes (PVC Data) to backup, where and what commands to execute, and how to react to some failures to achieve the best results from data snapshot functionality. You can select which volumes mounted at the Pod you want to backup, the preferred backup mode for the Pod, and what commands you want to execute.

The supported annotations are:

`bacula/backup.mode:[snapshot|standard]` defines how to handle PVC Data backups. If not defined, the default is `snapshot`.

`bacula/backup.volumes:<pvcname[,pvcname2...]>` defines what volumes will be backed up from this Pod. Multiple PVC names may be selected as a comma separated list. This annotation is required if you want to backup volumes on this Pod. Any other annotations are optional.

`bacula/run.before.job.container.command:<container>/<command>` defines what command to execute on which container of the Pod before any snapshot and data backup of this Pod occurs. An asterisk (`*`) as `<container>` name means to execute `<command>` on all containers.

`bacula/run.before.job.failjobonerror:[yes|no]` defines whether or not to fail the job when the exit code of the executed command is not zero. The default is yes.

`bacula/run.after.job.container.command:<container>/<command>` defines what command to execute on which container of the Pod after all snapshot and data backup of this Pod. An asterisk (`*`) as `<container>` name means to execute `<command>` on all containers.

`bacula/run.after.job.failjobonerror:[yes|no]` defines whether or not to fail the job when exit code of the executed command is not zero. The default is no.

`bacula/run.after.snapshot.container.command:<container>/<command>` defines what command to execute on which container of the Pod after all snapshot creations and

just before any data backup. An asterisk (*) as <container> name means to execute <command> on all containers.

bacula/run.after.snapshot.failjobonerror:[yes|no] defines to fail the job when exit code of the executed command is not zero. The default is no.

Pod annotations is an extension to the current PVC Data backup feature available with the pvcdata=... plugin parameter as described in 48.2.5 on page 524. This is an independent function which may be used together with the functionality described above, especially since both use the same data archive stream handling with Bacula Backup Pod.

All you need to use a new feature is to configure selected Pod annotations and make sure that the backup for a required Kubernetes namespace is properly configured. There is no need for any plugin configuration modifications. A Pod's volumes will be backed up automatically.

48.1.7 Examples

Below you can find some examples how to configure Bacula annotations in Kubernetes Pods.

In the example below you will use a simple Linux command sync to synchronize cached writes to persistent storage before volume snapshot.

```
apiVersion: v1
  kind: Pod
  metadata:
    name: app1
    namespace: default
    annotations:
      bacula/backup.mode: snapshot
      bacula/run.before.job.container.command: "*/sync -f /data1; sync -f /data2"
      bacula/run.before.job.failjobonerror: "no"
      bacula/backup.volumes: "pvc1,  pvc2"
  spec:
    containers:
    - image: ubuntu:latest
      name: test-container
      volumeMounts:
      - name: pvc1
        mountPath: /data1
      - name: pvc2
        mountPath: /data2
      volumes:
      - name: pvc1
        persistentVolumeClaim:
          claimName: pvc1
      - name: pvc2
        persistentVolumeClaim:
          claimName: pvc2
```

In the example below you will use PostgreSQL's database data files quiesce feature to perform consistent backup with snapshot[2].

The first command (*run.before.job.container.command*) freezes writes to database files and the second (*run.after.snapshot.container.command*) resumes standard database operation as soon as PVC snapshot becomes ready.

[2]The final PostgreSQL backup solution requires more configuration and preparation which was skipped in this example to make it clear

```
apiVersion: v1
kind: Pod
metadata:
  name: postgresql13
  namespace: default
  annotations:
    bacula/backup.mode: standard
    bacula/run.before.job.container.command: "*//bin/startpgsqlbackup.sh"
    bacula/run.after.snapshot.container.command: "*//bin/stoppgsqlbackup.sh"
    bacula/run.after.snapshot.failjobonerror: "yes"
    bacula/backup.volumes: "pgsql"
  spec:
    containers:
    - image: postgresql:13
      name: postgresql-server
      volumeMounts:
      - name: pgsql
        mountPath: /var/lib/pgsql
      volumes:
      - name: pgsql
        persistentVolumeClaim:
          claimName: pgsql-volume
```

48.1.8 Run Container Command annotation

All flavors of the Run Container Command parameters are remotely executed using the Kubernetes Pod remote execution API. Every command is prepared to execute with a standard Linux shell `/bin/sh`. This requires that a container image has to have the specified shell available. Using command shell execution gives flexibility to command execution or even allows for preparation of small scripts without additional container image customization.

48.1.9 Build

To enable the build of the Kubernetes plugin, add the following options to your `./configure` command line:

```
./configure --enable-kubernetes-plugin [... other options]
```

The Kubernetes plugin is written in Python, to ensure the stability of the Python environment, the default installation process will use Cython to convert the Python program to an executable.

48.1.10 Installation

The Bacula File Daemon and the Kubernetes plugin can be installed outside of the Kubernetes cluster on a server which has access to the Kubernetes API, or inside a protected cluster in a Container / Pod. The Kubernetes plugin can be installed on different operating systems and distributions, so the Bacula File Daemon for the correct operating system and platform has to be used.

There is no need, or in some solutions (when `K8S` is a cloud service like `GKE` or `EKS`), is it even a possible to install a Bacula File Daemon and the Kubernetes Plugin on a Kubernetes Master Server (`etcd`, `control pane`).

The `-enable-kubernetes-plugin` command line option of the `./configure` script will allow to install the kubernetes plugin with `make install`

The following packages are required to build and use the kubernetes plugin:

- Cython (to bundel python dependencies)

Note that it should be possible to use directly python files rather than Cython.

48.1.11 File Daemon Configuration

The `Plugin Directory` directive of `File Daemon` resource in `/opt/bacula/etc/bacula-fd.conf` must point to where the `kubernetes-fd.so` plugin file is installed. The standard Bacula plugin directory is `/opt/bacula/plugins`

```
FileDaemon {
  Name = bacula-fd
  Plugin Directory = /opt/bacula/plugins
  ...
}
```

48.1.12 Bacula Backup Proxy Pod Image

For Kubernetes PVC Data backup functionality, the Bacula Kubernetes plugin requires a dedicated Bacula Backup Proxy Pod which is automatically deployed using an image that is available in the bacula/scripts/kubernetes-bacula-backup directory.

This image should be installed manually on your local Docker images registry service which is available on your Kubernetes cluster as a source for application images.

Installation of the image can be performed with the following example commands:

```
# cd bacula-13.0.0/scripts/kubernetes-bacula-backup
# make
# docker load -i bacula-backup-<timestamp>.tar
# docker image tag bacula-backup:<timestamp> <registry>/bacula-backup:<timestamp>
# docker push <registry>/bacula-backup:<timestamp>
```

Where `<timestamp>` is the image version generated with the above package and `<registry>` is the location of your Docker images registry service. The exact procedure depends on your Kubernetes cluster deployment, so please verify the above steps before attempting to run the docker commands.

You can use any registry service available for your cluster, public or private, i.e. `gcr.io/`.

Depending on your cluster configuration it may be necessary to set the `baculaimage=<name>` plugin parameter (described at 48.2.6 on page 526) to define which repository and container image to use. The default for this parameter is `bacula-backup:<timestamp>` which may not be correct for your deployment.

Another example where you will need to modify the Bacula Backup Proxy Pod Image is in the case where your registry requires authentication. Please see the section 48.3.4 on page 535 for more details.

48.2 Backup and Restore Operations

48.2.1 Backup

The plugin can backup a number of Kubernetes Resources including: Deployments, Pods, Services or Persistent Volume Claims, check chapter 48.1.3 on page 516 for a complete list.

The backup will create a single (`.yaml`) file for any Kubernetes Resource which is saved. For PVC Data backup functionality the Kubernetes plugin generates a data archive as a single `<pvcname>.tar` archive file. The resources are organized inside the Bacula catalog to facilitate browsing and restore operations. In the Bacula catalog, Kubernetes resources are represented as follows:

- `/@kubernetes/namespaces/<namespace>.yaml` - Namespace definition
- `/@kubernetes/namespaces/<namespace>/<resource>/<name>.yaml` - Resource definitions in the namespace
- `/@kubernetes/namespaces/<namespace>/persistentvolumeclaim/<pvcname>.tar` - PVC Data backup in the selected namespace
- `/@kubernetes/persistentvolumes/<pvname>.yaml` - Persistent Volume definition
- `/@kubernetes/storageclass/<scname>.yaml` - Storage Class definition

All supported Kubernetes Resources will be saved if no filter options are set. You may limit which resources are saved using filtering options described in chapter 48.2.5 on page 524. By default, if no filter options are set, all supported Kubernetes Resources will be saved. To see the Kubernetes Resources that may be filtered, a listing mode is available. This mode is described in chapter 48.3.3 on page 534.

48.2.2 Restore

The Kubernetes plugin provides two targets for restore operations:

- Restore to a Kubernetes cluster
- Restore to a local directory

Restore to a Kubernetes Cluster

To use this restore method, the `where=/` parameter of a Bacula `restore` command is used. You can select any supported Kubernetes Resource to restore, or batch restore the whole namespace or even multiple namespaces. If you select a single resource to restore it will be restored as is without any dependent objects. In most cases, for (`Config Maps`, `Secrets`, `Services`, etc.) this is fine and restore will always succeed. On the other hand, compound objects (`Pods`, `Deployments`, etc.) won't be ready unless all dependencies are resolved during the restore. In this case you should make sure that you select all required resources to restore.

In Kubernetes, a successful resource restore doesn't necessarily result in the service successfully coming online. In some cases further monitoring and investigation will be required. For example:

- *Container image is unavailable.*
- *Volume Claim cannot provision new volume.*
- *Untrusted Image Repository.*
- *Infrastructure limits exceeded, i.e. a number of Pods or Memory allocations.*

- *etc. . .*

All example cases above must be resolved by the Kubernetes administrator. When all issues are resolved, the resource should automatically come online. If not, it may be necessary to repeat a restore to redeploy the Resource configuration.

The Kubernetes plugin does not wait for a Resource to come online during restore. It checks the Resource creation or replace operation status and reports any errors in the job log. The only exception to this is PVC Data restore, when the Kubernetes plugin will wait for a successful archive data restore. This operation is always executed at the end of the namespace recovery (when pvcdata is restored with other k8s objects) and should wait for proper PVC mount.

Restore to a Local Directory

To use this mode, the `where=/some/path` Bacula `restore` parameter is set to a full path on a server where the Bacula File Daemon and Kubernetes plugin are installed. If the path does not exist, it will be created by the Kubernetes plugin. With this restore mode you can restore any saved Kubernetes Resource including PVC Data archive file to a location on disk.

48.2.3 Plugin Configuration

The plugin is configured using `Plugin Parameters` defined in a `FileSets -> Include` section of the Bacula Enterprise Edition Director configuration.

48.2.4 Generic Plugin Parameters

The following Kubernetes plugin parameters affect any type of Job (Backup, Estimate, or Restore).

`abort_on_error[=<0 or 1>]` specifies whether or not the plugin should abort execution (and the Bacula Job) if a fatal error occurs during a Backup, Estimate, or Restore operation.

This parameter is optional.

`config=</path/to/file>` points to the location of the config file which defines how to connect to the Kubernetes cluster. You can read more about this *kubeconfig* file at: https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/ If this directive is omitted and no other access method is configured then a default config file location will be used - `$HOME/.kube/config`.

This parameter is optional.

`incluster` if this directive is defined then a standard in-cluster access to the Kubernetes API will be used. This option should be used only when the Bacula File Daemon and Kubernetes plugin are deployed as a Kubernetes cluster service in a Pod. In this case, Kubernetes itself will provide the required access credentials. This option won't work when executed outside of a Kubernetes cluster and services.

This parameter is optional.

`host=<url-k8s-api>` defines a Kubernetes API url. This option is used only with `token` parameter described below. If this option is specified, then both parameters are required.

This parameter is optional.

`token=<bearer-token>` defines a `Bearertoken` used for authorization to the Kubernetes API available at `host=<url-k8s-api>`. This option is used only with `host` parameter described above. You can read more about this type of authentication at: https://swagger.io/docs/specification/authentication/bearer-authentication/

This parameter is optional.

`verify_ssl[=<0 or 1>]` specifies whether or not the plugin should verify a Kubernetes API certificate when the connection uses SSL/TLS. If set to `verify_ssl=0` then verification will be disabled. This is useful when connecting to a Kubernetes API server with a self-signed certificate. The default behavior if this parameter if not set is to perform proper certificate validation.

This parameter is optional.

`ssl_ca_cert=</path/to/file>` specifies a file with the CA certificate used to customize the Kubernetes API server identity to verify the peer.

This parameter is optional.

`timeout=<seconds>` specifies the number of seconds for various network operations. Examples include: Waiting for Bacula Backup Proxy Pod connection or Kubernetes API operations, waiting for Pod execution or removal. The default is 600 seconds. The minimum timeout you may set is 1 second. When an invalid value is set the default will be used.

This parameter is optional.

`debug[=1]` specifies that the plugin backend will generate an execution debug file at location `/bacula/working/backendplugin/`. This file can help with troubleshooting the job execution if something goes wrong. If not defined then no debug file will be generated.

This parameter is optional.

48.2.5 Estimate and Backup Plugin Parameters

These plugin parameters are relevant only for Backup and Estimate jobs:

`namespace=<name>` specifies a Kubernetes namespace name which you want to backup. Multiple `namespace=<name>` parameters are allowed if you want to backup multiple namespaces. If a namespace with `name` can not be found its backup will be silently ignored. If this parameter is not set, all namespaces will be saved.

This parameter is optional.

`persistentvolume=<name>` specifies a Kubernetes persistent volume configuration you want to backup. Multiple `persistentvolume=<name>` parameters are allowed if you want to backup multiple volumes. You can use standard shell wildcard pattern matching to select multiple volumes using a single `persistentvolume` parameter. If a persistent volume with `name` can not be found its backup will be silently ignored. If this parameter is not set, all persistent volume configurations will be saved unless `pvconfig=0` is set as described below.

This parameter is optional.

`pvconfig=[0|1]` this option is used to disable persistent volume configuration backups when `pvconfig=0` is set. The default is to backup persistent volume configurations.

This parameter is optional.

`storageclass=<name>` specifies a Kubernetes Storage Class configuration to be backed up. Multiple `storageclass=<name>` parameters are allowed if you want to backup multiple resources. You can use standard shell wildcard pattern matching to select multiple volumes using a single `storageclass` parameter. If a storage class with `name` can not be found, its backup will be silently ignored. If this parameter is not set, all storage class configuration will be saved unless `scconfig=0` is set as described below.

This parameter is optional.

`scconfig=[0|1]` this option is used to disable Storage Class configuration backups when `scconfig=0` is set. The default is to backup Storage Class resource configurations.

This parameter is optional.

`pvcdata[=<pvcname>]` specifies a Kubernetes Persistent Volume Claim name you want to make a PVC Data archive for. As all PVCs are namespaced objects, to use this option you should select a required namespace with `namespace=<name>` parameter. If you define a simple `pvcdata` parameter without the equals sign ("=") and subsequent value, all detected persistent volume claims will be selected for the PVC Data archive backup.

This parameter is optional.

If none of the parameters above are specified, then all available Namespaces and Persistent Volume Configurations will be backed up. However, the plugin does not force a PVC Data archive backup in this case.

48.2.6 Backup and Restore Plugin Parameters

These plugin parameters are relevant only for Backup and Restore jobs when the PVC Data archive functionality is used:

`fdaddress=<IP or name>` is the IP address or host name where the Kubernetes plugin should listen for incoming connections from a Bacula Backup Proxy Pod. This parameter, combined with `fdport=<number>` below will define a socket to listen on. The default is to listen on all available interfaces (`0.0.0.0`) which should be fine in most cases.

This parameter is optional if `pluginhost=<IP or name>` is defined.

`fdport=<number>` is a port number on which Kubernetes plugin should listen for incoming connections from a Bacula Backup Proxy Pod. This parameter, combined with `fdaddress=<IP or name>` above will define a socket to listen on. The default is to listen on port `9104`.

This parameter is optional.

`pluginhost=<IP or name>` is the entry point address where a Bacula Backup Proxy Pod should connect during backup or restore operations. The name should be resolvable on the Kubernetes cluster, otherwise an IP address must be used here. This with `pluginport=<number>` parameter below will define the exact service entry point. The default is to use the value from `fdaddress=<IP or name>` parameter above.

This parameter is required when `fdaddress=<IP or name>` is not defined. Otherwise it is optional.

`pluginport=<number>` is the port number for service entry point address where the Bacula Backup Proxy Pod should connect during backup or restore operations. This, combined with the `pluginhost=<IP or name>` parameter above define the exact service entry point. The default is to use the value from the `fdaddress=<IP or name>` parameter above. When neither is defined the default `9104` port number will be used.

This parameter is optional.

`fdcertfile=<path>` is the SSL certificate file location for the Kubernetes plugin endpoint service data connection for a Bacula Backup Proxy Pod. The certificate and key (see `fdkeyfile=<path>` below) files are required for proper Bacula Backup Proxy Pod operations. You can create and use any valid SSL certificate including a self-signed one. You can even just use the certificate generated during the Kubernetes plugin installation located at `/opt/bacula/etc/bacula-backup.cert` The default is to use the certificate generated during plugin installation.

This parameter is optional.

`fdkeyfile=<path>` is an SSL private key file location for the SSL certificate defined by `fdcertfile=<path>` above. An unencrypted private key must be used for this purpose. The default is to use a private key file created during plugin installation at

/opt/bacula/etc/bacula-backup.key when fdcertfile=<path> above is not defined (the default). Otherwise the plugin will refer to the same certificate file from fdcertfile=<path> and use it as a .pem container.

This parameter is optional.

baculaimage=<name> is a Bacula Backup Proxy Pod container image registry location for your cluster as described in the image installation chapter 48.1.12 on page 521. In most cases this parameter will consist of your Docker images registry location with the tag of the required image. The default for this parameter is bacula-backup:<timestamp> which may not match your cluster configuration.

This parameter is optional.

imagepullpolicy=<Always|Never|ifNotPresent> sets the Kubernetes Pod image pull policy during Bacula Backup Proxy Pod container deployment. You can configure Kubernetes to Always or Never pull the image from the repository, or pull it only when not available with ifNotPresent. The option value is case insensitive. If this parameter is not defined, the default ifNotPresent will be used.

This parameter is optional.

48.2.7 Restore Plugin Parameters

During restore, the Kubernetes plugin will use the same parameters which were set for the backup job and saved in the catalog. During restore, you may change any of the parameters described in chapter 48.2.4 on page 523 and 48.2.6 on the previous page. In addition to the options used for backups, the outputformat option can be used during restore. This option specifies the file format when restoring to a local filesystem. You can choose the restore output in JSON or YAML. If not defined the restored files will be saved in YAML.

outputformat: <JSON or YAML> specifies the file format when restoring to a local filesystem as described above.

This parameter is optional.

48.2.8 FileSet Examples

In the example below, all Kubernetes Namespaces, Resources and Persistent Volume Configurations will be backed up using the default Kubernetes API access credentials.

```
FileSet {
  Name = FS_Kubernetes_All
  Include {
    Plugin = "kubernetes:"
  }
}
```

In this example, we will backup a single Kubernetes Namespace using the Bearer Token authorization method.

```
FileSet {
  Name = FS_Kubernetes_plugintest
  Include {
    Plugin = "kubernetes: host=http://10.0.0.1/k8s/clusters/test \
        token=kubeconfig-user:cbhssdxq8vv8hrcw8jdxs2 namespace=plugintest"
  }
}
```

The same example as above, but with a Persistent Volume:

```
FileSet {
  Name = FS_Kubernetes_mcache1
  Include {
    Plugin = "kubernetes: host=http://10.0.0.1/k8s/clusters/test \
        token=kubeconfig-user:cbhssdxq8vv8hrcw8jdxs2 \
        namespace=plugintest persistentvolume=myvol"
  }
}
```

This example backs up a single Namespace and all detected PVCs in this Namespace using a defined listening and entry point address and the default connection port:

```
FileSet {
  Name = FS_Kubernetes_test_namespace
  Include {
    Plugin = "kubernetes: namespace=test pvcdata fdaddress=10.0.10.10"
  }
}
```

The same example as above, but using different listening and entry point addresses as may be found when the service is behind a firewall using port forwarding features:

```
FileSet {
  Name = FS_Kubernetes_test_namespace_through_firewall
  Include {
    Plugin = "kubernetes: namespace=test pvcdata=plugin-storage fdaddress=10.0.10.10 \
        pluginhost=backup.example.com pluginport=8080"
  }
}
```

This example shows PVC Data archive backup with the Bacula File Daemon inside a Kubernetes cluster:

```
FileSet {
  Name = FS_Kubernetes_incluster
  Include {
    Plugin = "kubernetes: incluster namespace=test pvcdata \
        pluginhost=backup.bacula.svc.cluster.local"
  }
}
```

The configuration above is designed for use in situations where the Bacula server components are located on-premise and behind a firewall with no external ports allowed in, but must back up data on an external Kubernetes cluster.

48.3 Restore examples

48.3.1 Restore to Kubernetes Cluster

To restore Kubernetes resources to a Kubernetes cluster, the administrator should execute the restore command and specify the `where` parameter as in this example:

```
* restore where=/
```

and then set any other required restore plugin parameters for the restore.

```
* restore where=/
...
$ cd /@kubernetes/namespaces/plugintest/configmaps/
cwd is: /@kubernetes/namespaces/plugintest/configmaps/
$ ls
plugintest-configmap.yaml
$ add *
1 file marked.
$ done
Bootstrap records written to /opt/bacula/working/bacula-devel-dir.restore.1.bsr

The Job will require the following (*=>InChanger):
Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

Vol005                    File1                     FileChgr1

Volumes marked with "*" are in the Autochanger.


1 file selected to be restored.

Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-devel-dir.restore.1.bsr
Where:           /
Replace:         Always
FileSet:         Full Set
Backup Client:   bacula-devel-fd
Restore Client:  bacula-devel-fd
Storage:         File1
When:            2019-09-30 12:39:13
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): mod
Parameters to modify:
1: Level
2: Storage
3: Job
4: FileSet
5: Restore Client
6: When
7: Priority
8: Bootstrap
9: Where
```

```
10: File Relocation
11: Replace
12: JobId
13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: config=/home/radekk/.kube/config
Plugin Restore Options
config:              radekk/.kube/config  (*None*)
host:                *None*               (*None*)
token:               *None*               (*None*)
username:            *None*               (*None*)
password:            *None*               (*None*)
verify_ssl:          *None*               (True)
ssl_ca_cert:         *None*               (*None*)
outputformat:        *None*               (RAW)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
1: config (K8S config file)
2: host (K8S API server URL/Host)
3: token (K8S Bearertoken)
4: verify_ssl (K8S API server cert verification)
5: ssl_ca_cert (Custom CA Certs file to use)
6: outputformat (Output format when saving to file (JSON, YAML))
7: fdaddress (The address for listen to incoming backup pod data)
8: fdport (The port for opening socket for listen)
9: pluginhost (The endpoint address for backup pod to connect)
10: pluginport (The endpoint port to connect)
Select parameter to modify (1-8): 1
Please enter a value for config: /root/.kube/config
Plugin Restore Options
config:              /root/.kube/config   (*None*)
host:                *None*               (*None*)
token:               *None*               (*None*)
verify_ssl:          *None*               (True)
ssl_ca_cert:         *None*               (*None*)
outputformat:        *None*               (RAW)
fdaddress:           *None*               (*FDAddress*)
fdport:              *None*               (9104)
pluginhost:          *None*               (*FDAddress*)
pluginport:          *None*               (9104)
Use above plugin configuration? (yes/mod/no): yes
Job queued. JobId=1084
\end{verbatimx}

The plugin does not wait for Kubernetes Resources to become ready and online in the same way
as the \texttt{kubectl} or the \texttt{oc} commands.

\subsection{Restore to a Local Directory}
\label{localdirrestore}
It is possible to restore any Kubernetes Resource(s) to file without
loading them into a cluster. To do so, the \texttt{where} restore option should point
to the local directory:

\begin{verbatim}
* restore where=/tmp/bacula/restores
...
$ cd /@kubernetes/namespaces/
cwd is: /@kubernetes/namespaces/
$ ls
```

```
bacula/
cattle-system/
default/
graphite/
ingress/
plugintest/
$ add plugintest
25 files marked.
$ done
Bootstrap records written to /opt/bacula/working/bacula-devel-dir.restore.2.bsr

The Job will require the following (*=>InChanger):
Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

Vol005                    File1                     FileChgr1

Volumes marked with "*" are in the Autochanger.


25 files selected to be restored.

Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-devel-dir.restore.2.bsr
Where:           /tmp/bacula/restores
Replace:         Always
FileSet:         Full Set
Backup Client:   bacula-devel-fd
Restore Client:  bacula-devel-fd
Storage:         File1
When:            2019-09-30 12:58:16
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): mod
Parameters to modify:
1: Level
2: Storage
3: Job
4: FileSet
5: Restore Client
6: When
7: Priority
8: Bootstrap
9: Where
10: File Relocation
11: Replace
12: JobId
13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: config=/home/radekk/.kube/config debug=1
Plugin Restore Options
config:              *None*              (*None*)
host:                *None*              (*None*)
token:               *None*              (*None*)
verify_ssl:          *None*              (True)
ssl_ca_cert:         *None*              (*None*)
outputformat:        *None*              (RAW)
```

```
fdaddress:           *None*               (*FDAddress*)
fdport:              *None*               (9104)
pluginhost:          *None*               (*FDAddress*)
pluginport:          *None*               (9104)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
1: config (K8S config file)
2: host (K8S API server URL/Host)
3: token (K8S Bearertoken)
4: verify_ssl (K8S API server cert verification)
5: ssl_ca_cert (Custom CA Certs file to use)
6: outputformat (Output format when saving to file (JSON, YAML))
7: fdaddress (The address for listen to incoming backup pod data)
8: fdport (The port for opening socket for listen)
9: pluginhost (The endpoint address for backup pod to connect)
10: pluginport (The endpoint port to connect)
Select parameter to modify (1-8): 8
Please enter a value for outputformat: JSON
Plugin Restore Options
config:              *None*               (*None*)
host:                *None*               (*None*)
token:               *None*               (*None*)
verify_ssl:          *None*               (True)
ssl_ca_cert:         *None*               (*None*)
outputformat:        *None*               (RAW)
fdaddress:           *None*               (*FDAddress*)
fdport:              *None*               (9104)
pluginhost:          *None*               (*FDAddress*)
pluginport:          JSON                 (9104)
Use above plugin configuration? (yes/mod/no): yes
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-devel-dir.restore.2.bsr
Where:           /tmp/bacula/restores
Replace:         Always
FileSet:         Full Set
Backup Client:   bacula-devel-fd
Restore Client:  bacula-devel-fd
Storage:         File1
When:            2019-09-30 12:58:16
Catalog:         MyCatalog
Priority:        10
Plugin Options:  User specified
OK to run? (yes/mod/no):
Job queued. JobId=1085
```

Output format conversion at restore time will format all data in a human readable format. You can find an example of this restore below.

```
# cat /tmp/bacula/restores/namespaces/plugintest/plugintest.json
{
"apiVersion": "v1",
"kind": "Namespace",
"metadata": {
"annotations": {
"field.cattle.io/projectId": "c-hb9ls:p-bm6cw",
"lifecycle.cattle.io/create.namespace-auth": "true"
},
"cluster_name": null,
```

```
"creation_timestamp": "2019-09-25T16:31:03",
"deletion_grace_period_seconds": null,
"deletion_timestamp": null,
"finalizers": [
"controller.cattle.io/namespace-auth"
],
"generate_name": null,
"generation": null,
"initializers": null,
"labels": {
"field.cattle.io/projectId": "p-bm6cw"
},
"name": "plugintest",
"namespace": null,
"owner_references": null,
"resource_version": "11622",
"self_link": "/api/v1/namespaces/plugintest",
"uid": "dd873930-dfb1-11e9-aad0-022014368e80"
},
"spec": {
"finalizers": [
"kubernetes"
]
},
"status": {
"phase": "Active"
}
}
```


The supported output transformations are: `JSON` and `YAML`.

48.3.2 Restore PVC Data Archive

Here we describe functionalities and requirements related to pvcdata restore.

Local Directory Restore

The procedure to restore a PVC Data archive file to a local directory is basically the same as restoring the Kubernetes Resource configuration file as described in **??** on page ??. However, output transformation is unavailable and ignored when restoring PVC data. Restore of this data will create a `tar` archive file you can manually inspect and use.

Restore to PVC

This procedure is similar to the one described in PVC Data backup and uses the same Bacula Backup Proxy Pod image. During restore, the plugin uses the same endpoint configuration parameters so it is not necessary to setup it again. If your endpoint parameters have changed you can update them using Bacula plugin restore options modification as in example below:

```
*restore select all done where=/
(...)
OK to run? (yes/mod/no): mod
Parameters to modify:
1: Level
```

```
2: Storage
3: Job
4: FileSet
5: Restore Client
6: When
7: Priority
8: Bootstrap
9: Where
10: File Relocation
11: Replace
12: JobId
13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: namespace=plugintest pvcdata pluginhost=example.com
Plugin Restore Options
config:              *None*               (*None*)
host:                *None*               (*None*)
token:               *None*               (*None*)
verify_ssl:          *None*               (True)
ssl_ca_cert:         *None*               (*None*)
outputformat:        *None*               (RAW)
fdaddress:           *None*               (*FDAddress*)
fdport:              *None*               (9104)
pluginhost:          *None*               (*FDAddress*)
pluginport:          *None*               (9104)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
1: config (K8S config file)
2: host (K8S API server URL/Host)
3: token (K8S Bearertoken)
4: verify_ssl (K8S API server cert verification)
5: ssl_ca_cert (Custom CA Certs file to use)
6: outputformat (Output format when saving to file (JSON, YAML))
7: fdaddress (The address for listen to incoming backup pod data)
8: fdport (The port for opening socket for listen)
9: pluginhost (The endpoint address for backup pod to connect)
10: pluginport (The endpoint port to connect)
Select parameter to modify (1-10): 9
Please enter a value for pluginhost: newbackup.example.com
Plugin Restore Options
config:              *None*               (*None*)
host:                *None*               (*None*)
token:               *None*               (*None*)
verify_ssl:          *None*               (True)
ssl_ca_cert:         *None*               (*None*)
outputformat:        *None*               (RAW)
fdaddress:           *None*               (*FDAddress*)
fdport:              *None*               (9104)
pluginhost:          newbackup.example.com (*FDAddress*)
pluginport:          *None*               (9104)
Use above plugin configuration? (yes/mod/no): yes
```

You can restore all data available from the backup archive for a selected Persistent Volume Claim and all data will be overwritten, ignoring the `Replace` job parameter. Please take note of this behavior, which may change in the future.

48.3.3 Other

Resource listing

The Bacula Enterprise Kubernetes plugin supports the "plugin listing" feature of Bacula Enterprise Edition 8.x or newer. This mode allows the plugin to display some useful information about available Kubernetes resources such as:

- List of Kubernetes namespaces
- List of Kubernetes persistent volumes
- List of Kubernetes storage class resources

The feature uses the special `.ls` command with a `plugin=<plugin>` parameter.

The command requires the following parameters to be set:

`client=<client>` A Bacula Client name with the Kubernetes plugin installed.

`plugin=<plugin>` A plugin name, which would be `kubernetes:` in this case, with optional plugin parameters as described in section 48.2.4 on page 523.

`path=<path>` An object path to display.

The supported values for a `path=<path>` parameter are:

`/` to display Object types available to list

`namespaces` to display a list of Namespaces

`persistentvolumes` to display a list of Persistent Volumes

`storageclass` to display a list of Storage Class Resources

`namespaces/<name>/pvcdata` to display all available Persistent Volume Claims available in Namespace `<name>`

To display available Object types, follow the following command example:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/
Connecting to Client kubernetes-fd at localhost:9102
drwxr-x---   1 root     root 2018-09-28 14:32:20  /namespaces
drwxr-x---   1 root     root 2018-09-28 14:32:20  /persistentvolumes
drwxr-x---   1 root     root 2018-09-28 14:32:20  /storageclass
2000 OK estimate files=2 bytes=0
```

To display the list of all available Kubernetes namespaces, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=namespaces
Connecting to Client kubernetes-fd at localhost:9102
drwxr-xr-x   1 root     root 2019-09-25 16:39:56  /namespaces/default
drwxr-xr-x   1 root     root 2019-09-25 16:39:56  /namespaces/kube-public
drwxr-xr-x   1 root     root 2019-09-25 16:39:56  /namespaces/kube-system
drwxr-xr-x   1 root     root 2019-09-25 16:46:19  /namespaces/cattle-system
drwxr-xr-x   1 root     root 2019-09-27 13:04:01  /namespaces/plugintest
2000 OK estimate files=5 bytes=0
```

To display the list of available Persistent Volume Claims which could be used for PVC Data archive feature selection, you can use the following example command for the mysql namespace:

```
*.ls client=bacula-devel-fd plugin="kubernetes:" path=/namespaces/mysql/pvcdata
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root 2019-10-16 14:29:38  /namespaces/mysql/pvcdata/mysql-mysql
2000 OK estimate files=1 bytes=0
```

To display the list of all available Persistent Volumes, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=persistentvolumes
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-bfaebd0d-dfad-11e9-a2cc-42010a8e017
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-b1a49497-dfad-11e9-a2cc-42010a8e017
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-949cb638-dfad-11e9-a2cc-42010a8e017
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-9313388c-dfad-11e9-a2cc-42010a8e017
-rw-r----- 10737418240 2019-09-24 /persistentvolumes/myvolume
2000 OK estimate files=5 bytes=15,032,385,536
```

The volume lists display a Volume Storage size which does not reflect the actual configuration size during backup.

To display the list of all defined Storage Class Resources, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=storageclass
Connecting to Client kubernetes-fd at localhost:9102
-rw-r----- 1024 2020-07-27 13:39:48  /storageclass/local-storage
-rw-r----- 1024 2020-07-23 16:14:13  /storageclass/default-postgresql-1
-rw-r----- 1024 2020-07-24 11:47:02  /storageclass/local-storage-default
-rw-r----- 1024 2020-07-23 12:00:02  /storageclass/standard
2000 OK estimate files=4 bytes=4,096
```

48.3.4 Advanced Bacula Backup Proxy Pod Deployment

WARNING: This is an advanced topic related to Kubernetes clusters. You should !!NOT!! try to implement or customize the Bacula Kubernetes plugin behavior unless you REALLY know what you are doing.

You can customize the service parameters used for deploying Bacula backup Pods dedicated to Persistent Volume Claim data backups to suit your needs. The plugin uses the following Pod service deployment YAML template to execute the proxy operation pod on the cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: {podname}
  namespace: {namespace}
  labels:
    app: {podname}
spec:
  hostname: {podname}
  {nodenameparam}
```

```
  containers:
  - name: {podname}
    resources:
      limits:
        cpu: "1"
        memory: "64Mi"
      requests:
        cpu: "100m"
        memory: "16Mi"
    image: {image}
    env:
    - name: PLUGINMODE
      value: "{mode}"
    - name: PLUGINHOST
      value: "{host}"
    - name: PLUGINPORT
      value: "{port}"
    - name: PLUGINTOKEN
      value: "{token}"
    imagePullPolicy: {imagepullpolicy}
    volumeMounts:
      - name: {podname}-storage
        mountPath: /{mode}
  restartPolicy: Never
  volumes:
    - name: {podname}-storage
      persistentVolumeClaim:
        claimName: {pvcname}
```

The above template uses a number of predefined placeholders which will be replaced by corresponding variables during Pod execution preparation. To customize proxy Pod deployment you can change or tune template variables or the template body. Below is a list of all supported variables with short descriptions and requirement conditions.

`podname` - This is the predefined Pod name used by a plugin. This variable is required and cannot be customized.

`namespace` - This is a Namespace name for which the PVC Data backup is performed. This variable is required and cannot be customized.

`nodenameparam` - This is a placeholder for Cluster node name parameter (`nodeName: ...`) used to mount an existing Volume Claim for backup or restore if required for the selected PVC. In most cases this variable is required and cannot be customized.

`image` - This is a Pod container image name to execute. You can customize or omit this variable as long as you provide a container image name required by the cluster.

`mode` - This is an operation mode for the Proxy Pod. The supported values are: `backup` and `restore`. This variable is required and cannot be customized.

`host` - This is an endpoint address which corresponds to the `pluginport=...` Kubernetes plugin parameter. This variable is required. You can customize or omit this variable as long as you provide a value for the `PLUGINHOST` container environment.

`port` - This is an endpoint address which corresponds to the `pluginport=...` Kubernetes plugin parameter. This variable is required. You can customize or omit this variable as long as you provide a value for the `PLUGINPORT` container environment.

`token` - This is an Authorization Token (randomly generated). This variable is required and cannot be customized.

pvcname - This is the name of the PVC for backup or restore operations. This variable is required and cannot be customized.

You can create the required file: /opt/bacula/scripts/bacula-backup.yaml or point to the custom one using $DEFAULTPODYAML environment variable.

48.3.5 Limitations

- Only full level backups are possible. This is a Kubernetes limitation.
- You can perform a single PVC Data backup or restore with a single Bacula File Daemon installation associated with single fdaddress=<name>. This limitation may be removed in a future release of the Kubernetes plugin.

48.3.6 Common Problems

In this section we describe the common problems you may encounter when you first deploy the Bacula Kubernetes plugin or when you are not very familiar with this plugin.

- You have set the incluster parameter in your Job but you have the following error:

  Error: kubernetes: incluster error: Service host/port is not set.

  This means you are running the Bacula File Daemon and Kubernetes plugin not in a Pod, or Kubernetes does not provide default service access in your installation. In the latter case you should use a standard Kubernetes access method in a prepared kubeconfig file.

Chapter 49

User Interfaces

Bacula provides several user interfaces with the core software distribution, and additional interfaces are available from developers only loosely connected to the community or Bacula Systems.

The core interfaces are the terminal / console program `bconsole` and the Tray Monitor program that is intended to provided limited functionality, but integrate into graphical user interfaces.

In addition, several web interfaces do exist; the most prominent ones are

- BWeb
- Bacula-Web
- Baculum

49.1 The Tray Monitor

The Tray Monitor program is a simplified graphical user interface intended to be used with GUIs providing a tray area, similar to Windows' Task Bar information icon tray area.

It provides the capability to monitor all the core components of a Bacula infrastructure, namely Directors, Storage Daemons and File Daemons. The Tray Monitor program can be configured to monitor an arbitrary number of those daemons, and it can show an icon indicating current activity of the monitored daemon.

The Tray Monitor also has a built-in graphical configuration interface (but it can, as all Bacula components, be configured with a plain text file) and allows to initiate backup jobs through its context menu.

In addition, the Tray Monitor program can interactively or non-interactively start jobs.

The latter functionality is particularly valuable if used in conjunction with the capability of File Daemons to serve as proxy when connecting to a Director needing a connection to the client to run a job.

49.1.1 Installation

When building the Bacula software from source, the Tray Monitor program will be built similarly and under the same conditions as BAT, the Bacula Administration Tool, is. In particular, it needs a Qt build environment.

When installing the client software from packages, the Tray Monitor may be packaged individually or along with some other console programs – this is a packager's decision. Bacula Systems' Enterprise Edition packages the Tray Monitor together with the GUI console (BAT) on Linux, and with the client installer on Windows.

The actual steps to install depend on the platform and the packages used. A Windows Installer will typically allow to select the Tray Monitor for installation in some tree view of components to be installed.

The Tray Monitor program can be started whenever a GUI session is initiated, but care should be taken to use a configuration file that belongs to the user who runs the GUI session. In particular, having a default configuration file with credentials allowing unrestricted Director access usable by regular end users is to be avoided.

49.1.2 Configuration

The Tray Monitor program, unlike most other parts of the Bacula software, can be run without a configuration file in place. In this case, the configuration panel will be opened, allowing the user to configure the program according to her needs.

With the Tray Monitor it is quite likely that individual users will have their own configuration, i.e. a system-wide configuration file is not the common way to run the program. This is because the Tray Monitor, to do its work, needs access to backend components of the Bacula infrastructure, and individual users will only be provided with restricted access by the Bacula administration for security reasons.

For example, on a system shared by regular users and system administrators, the non-privileged users should not be able to see all jobs that can be run but only the ones relating to their own data, while a system administrator probably needs to monitor all the components running on the machines under his responsibility.

GUI Configuration

When started without a configuration file, or when the "Configure..." context menu item of the tray icon is clicked, the configuration window of the Tray Monitor program is shown. It consists of a tabbed view of configuration pages and some buttons, which provide the following functionality:

Save to save the current state into a text file (see paragraph above).

Cancel to abort the operation, leave everything as it was, and close the configuration panel.

Password to toggle between showing passwords on the current tab page in clear or hidden. This is useful to keep passwords hidden against observers, but still be able to check potentially complex and long passwords when needed.

(Add) Client to add a new, empty client configuration tab.

(Add) Storage to add a new, empty storage configuration tab.

(Add) Director to add a new, empty director configuration tab.

There will always be at least one tab to configure the identity of this Tray Monitor instance itself. Configuration page tabs that represent Clients, Storages, and Directors, will initially appear with a title of "New <Component>" which will change to the name of the monitored component once that has been provided.

The Monitor Configuration page

This tab defines the identity of the Tray Monitor instance. It is always available, cannot be removed, and there can only be one such tab. It contains the following fields:

Name the name of this Tray Monitor instance. This name is used to authenticate when connecting to any other Bacula daemon.

Note that, as usual with Bacula, the password used for authentication is configured individually for each counterpart.

The name is a *name-string* data type and allows only a limited set of characters (essentially, ASCII characters and numerals plus basic interpunction) and is limited to a length of 127 characters.

Refresh Interval is used to set the number of seconds between refreshes of status displays. Integer numbers between 5 and 9999 can be entered or selected.

Command Directory allows to enter or select the path of a directory which will be polled for command files. See below for details on how to use this feature.

Display Advanced Options will enable a settings page to select more Job options when the "Run..." context menu item of the Tray Monitor program is selected. See below for more information.

As usual for a Bacula component, unless the identity is configured the program will not be very functional.

Common elements

A common layout is used throughout all the configuration pages for the different Bacula Daemons. We will describe that first, before we look at specific additional configuration items for Client and Director.

We have always two groups of settings and a waste bin icon. The waste bin is used to delete the currently shown configuration from the Tray Monitor setup.

One group of settings is used to define the essential information to contact and use the component, and one group to configure TLS-related information. The former contains some required fields, among them the connection and authentication information, while the latter is optional.

We will not discuss the details of TLS configuration here – if TLS is required, the Bacula administrator will probably give detailed instructions and has to provide at least the "CA Certificate File". Simply put, leave the "Enabled" check box in the TLS section turned off unless detailed and site-specific configuration information is provided. If it is enabled, the selections in this group are critical and need to match what is configured for the respective daemon.

The General section contains the following elements common to Client, Storage, and Director configuration pages:

Name to configure the name that is used to represent this component in the Tray Monitor's user interface. This is a typical Bacula "name-string" with restrictions in length and usable character set (see above in the "Monitor Configuration" description).

Description can hold a textual description of the component. It is not used for any functionality, but rather to allow users to annotate their configuration.

The length of the description text is limited to 512 characters, and it can contain (nearly) any character as it's of the configuration data type "string".

Password is used to configure the shared secret that authenticates against the configured daemon. By default the text entered here is hidden, but it is possible to toggle with clear-text display using the "Password" button on the right.

The password is of Bacula's "password" data type and as such can contain up to 127 characters. ASCII symbols, numerals, and some punctuation are allowed here.

Address is the address at which the configured daemon can be contacted. Usually, an IPv4 address in dotted quad notation or a DNS name is used.

Note that domain names should be fully qualified – using plain host names is strongly discouraged.

Support for IPv6 addresses depends on how the Tray Monitor program was built, but should be matching the IPv6 support status of all other Bacula programs resulting from the same build run.

Port represents the port number at which the daemon that is configured to be monitored can be contacted. Integer numbers between 1 and 65535 are allowed here. The default value shown in a new configuration page is the default port number for the daemon type.

This value should only be modified in two cases:

1. When the Bacula administrator explicitly indicates that a specific, non standard port number should be used.
2. When the File Daemon is configured for client-initiated backup. In this case the address to use will usually be "localhost", the port 9102, and the "Remote" checkbox will be ticked.

   See below for more information on this mode of operation.

Timeout is used to set the connection timeout so the Tray Monitor will not try infinitely long to connect to the counterpart.

This is a typical Bacula "time-string", so the input format can be pretty flexible. However, for the purpose of network timeouts, plain integers representing numbers are suitable, and values between 5 and 300 (seconds) should be reasonable. The default unit is seconds.

Monitor is a checkbox which, when selected, tells the Tray Monitor program that the currently configured daemon should update the tray icon. When the daemon has some activity the tray icon will indicate this. Otherwise the tray icon will show an idle state.

This setting is most useful on client systems where it can give an indication of current activity to the desktop user, so that disrupting Bacula operations by shutting down or disconnecting from the network can be avoided.

Client configuration

In most cases, the Bacula File Daemon will be the one running on the local machine. In addition, a Client can also serve as a proxy for the Tray Monitor program to connect to a Director. This additional functionality is enabled with a configuration item:

Remote tells the Tray Monitor that this client can be used as a proxy to connect to a Director. This operation mode allows to initiate connections between Director and File Daemon from the client side, which is invaluable in situations where the DIR cannot (reliably) contact the File Daemon. Such a client-initiated connections can then be used to run jobs on clients inaccessible by the Director. See 49.1.6 on page 547 for more details.

Director Configuration

In addition to the common configuration settings described above, one additional parameter is available here:

Use SetIp causes the Tray Monitor program to send a `setip` command to the Director, which will enable the Director to communicate with this client.

This setting can be useful in cases where the client computer has no fixed IP address or DNS name, but can be reached directly by the Director. Once the `setip` command was executed, the Director can use regular ways to operate on this client, for example starting scheduled Jobs or querying the Client status.

For the `setip` command to succeed, it is required to use exactly identical Client and Console resource names in the Director's configuration. The Main manual describes this feature in detail.

Configuration file structure

As usual for Bacula components, the configuration for the Tray Monitor program is stored as a plain text file; the format is exactly the same as used with all the other components.

When you configure Bacula Tray Monitor with the GUI, a certain knowledge of the Bacula configuration scheme is surely helpful. When you edit the file with a text editor, this knowledge becomes essential.

However, there is a rather simple mapping from fields in the GUI to configuration file contents: Each tab represents a resource of the specified type, and each actual setting is named identical or similarly in both the GUI and the file. For example, the following configuration file

```
Monitor {
  Name="wgb-sql14b-con"
  Refresh Interval = 1001
```

```
  Display Advanced Options = yes
}
Client {
  Name = "wgb-sql14b-fd"
  Address = "localhost"
  Password = "85test-onremote"
  Port = 9102
  Connect Timeout = 10
  Remote = yes
  Monitor = yes
}
```

exactly represents the settings in the GUI as seen in figure 49.1.

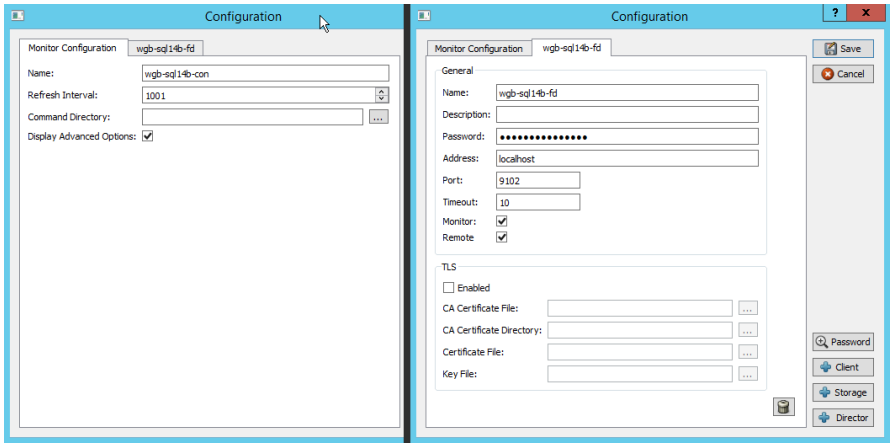


Figure 49.1: Tray Monitor Configuration as seen in Listing

Default locations of configuration files

The default locations of user specific configuration files that are written by the GUI are `/home/username/.bacula-tray-monitor.conf` (Linux) and `C:\Users\username\AppData\Roaming\bacula-tray-monitor.conf` (Windows, note that AppData is a hidden directory).

49.1.3 Monitoring

After double-clicking the Tray Monitor icon, or when selecting the "Display..." item in its context menu, the monitor screen is shown. In this screen, a tab selects which of the configured components of Bacula to show.

The monitoring panel consists of three areas, well known to any Bacula administrator, but probably not so easily understood by users without prior Bacula knowledge. On top, the general daemon status is shown. This gives information about the identity of the component, in particular the configured name, the version of the software, which plugins are currently loaded, and when the service process was started. In addition, some global settings may be shown – for a File Daemon, this is the configured Bandwidth Limit. Note that in-depth diagnostic information as presented by the `bconsole status` command is **not** shown.

Below the general status a list shows all currently running jobs. This list contains the JobID and the job's name, and as seen in figure 49.2 on the facing page, with the number of files and the amount of data already processed also gives some indication of job progress. The Error counter column, while looking important, is much less so than it appears, since actual error causes or messages can not be seen here.

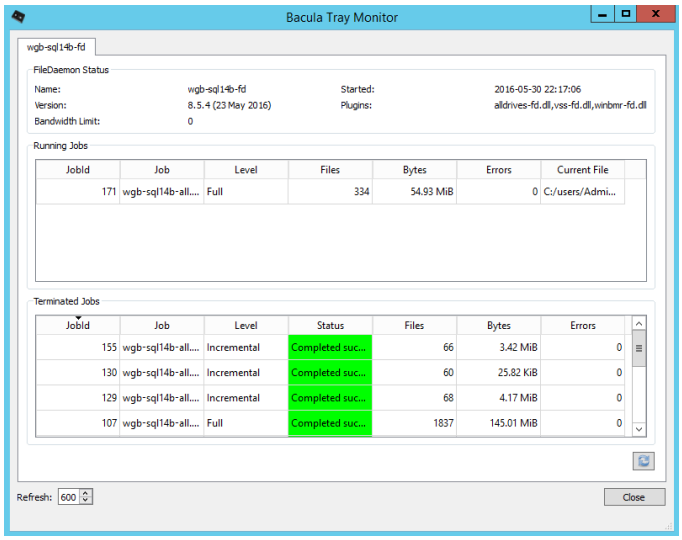


Figure 49.2: Tray Monitor Client Display

Finally, the lowermost part of the monitoring panel shows an overview of the last jobs run by the particular daemon. Status text field is color coded (e.g. green for successful jobs).

Note that column headers can be clicked to change the sort order of the table, and column widths can be modified by dragging the table header column separators.

In the bottom right corner of the monitoring panel there is an icon to immediately refresh the display, while outside of the monitoring panel and the tab view, in the bottom left corner of the display window, the automatic refresh interval can be adjusted.

The "Close" button at the bottom right corner will close the display window in the same way as a click on the window decoration close button does.

49.1.4 Running Jobs

You can start a job by selecting "Run..." from the context menu of the tray icon.

If more than one Director or Clients with "Remote" capability are configured, a window will be shown to select the Director to use for the Job to be run. This window will appear as shown in figure 49.3. The name used in the selection drop-down menu is the one configured in the "Name:" setting of the component, see above in the "Configuration" section.

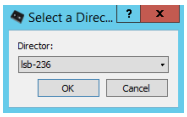


Figure 49.3: Selecting a Director to run a Job

The "Run Job" window will be shown, which presents Job properties and an estimate of how much data will be backed up. The estimate will be updated when Job properties are changed. Be aware that this estimate is not based on current backup source information, i.e. it is not the result that you would get with the `estimate` command, but it is derived using a simple regression from Catalog information. The information presented is, however, usually enough to determine if the Job can finish during the remaining working hours.

Depending on the general Monitor configuration setting to "Display Advanced Options", a second, tabbed panel will be available and allow to change many other Job parameters, such as the backup level, the File Set to use, or the backup target like Pool and Storage device.

All settings are subject to access control restrictions configured at the Director, thus it is possible to have specific and secure options per each user (with each user connecting to their "own" access controlled Console resource).

49.1.5 Local Scheduling – the Command Directory

The Tray Monitor configuration item `Command Directory` can be used to automate Bacula activities through the Tray Monitor itself. In particular it allows to schedule jobs on the client machine, and not (only) inside the Director's configuration.

If configured with a path readable by the Tray Monitor program, the Tray Monitor, while running, will regularly look for files ending on `".bcmd"` and process them. Once a file was processed, it will be renamed to end with `.bcmd.ok`. These command files need to follow a specific syntax:

```
component-name: bconsole-command
```

`component-name` must be one of the defined Director or Client resources (Client only with **Remote**=Yes).

`bconsole-command` is just a plain `bconsole` command which will be sent to the Director (possibly through the File Daemon).

The most useful command here is `run job=job-name` to initiate backups, where `job-name` would be the one that backs up from the local machine.

The solution to schedule backups of the local machine is accordingly to have `cron` or Windows' Task Scheduler create appropriate command files in the right location.

Part of that scheduled task could be to verify connectivity to the Director. An example suitable on a Linux or Unix system might look like this:

```
#!/bin/sh
if ping -c 1 director &> /dev/null
then
  echo "my-dir: run job=backup" > /path/to/commands/backup.bcmd
fi
```

On a Windows system, a batch script like the following could be used:

```
@ECHO OFF
ping -n 1 -4 lsb-236.lsb.os.baculasystems.com >nul:
if ERRORLEVEL 1 (
   echo Director not reachable!
) else (
   echo wgb-sql14b-fd: run job=wgb-sql14b-all>%LOCALAPPDATA%\Bacula\commands\runnow.bcmd
)
```

Using `cron` to start this sort of script is pretty simple for a user with some Linux or Unix knowledge. Unfortunately, creating a scheduled task on a Windows machine is somewhat difficult to explain due to the number of required mouse clicks. We may provide step-by-step instructions at a later time.

49.1.6 Proxied Connection

With the Tray Monitor program providing a user friendly interface, and support for local scheduling, there's only one element missing to enable Bacula backing up computers that are unreliably available and may not be accessible from the Director itself: A way to enable the Director to run a Job on a given client, even if, from the Director, the client can not be accessed.

For many desktop and most laptop users, Bacula until now was not a very convenient solution, because it relied on the Director being configured with the client computer's IP address or host name, and then the Director being able to connect to the client.

In many corporate networks, however, there is neither DNS resolution nor static IP addressing in place for desktop computers. Furthermore, if the client computer is connected outside of the organization's LAN, for example in a home office behind a NAT'ing router, there will be no reliable way to connect to it at all.

In those situations, it is essential that the File Daemon connects to the Director, and the Director then uses this TCP connection to control the backup Job.

This is possible as of version 8.6 of Bacula Enterprise Edition using the *Client-Initiated Connection* feature. Note that both Director and File Daemon need to support this functionality.

As we assume that such a setup of Bacula will usually be implemented to back up user computers, we will provide the documentation including a restricted console using ACLs.

We will first provide a description of the connection flow for a job that makes use of the proxied console connection, and then describe the required configuration in detail.

Connection Flow

For a Client-Initiated Backup to work through an FD-initiated connection, the following sequence of events will happen:

1. Console wants to start a Job, either manually or automated.
   The Director to run this job on should be the local FD, and, if the Tray Monitor is used, it should be configured with `Remote = Yes`.
2. The Console (most likely, the Tray Monitor program) connects to the local FD.
   Inside the FD configuration, such incoming connections are represented as a Director resource with two new settings: `Remote` and `Console`.
3. The Console submits the `proxy` command to the FD.
   Following this, the FD will use the Console resource referenced by the Director resource the current Console connection uses and use the provided information (Address, Name, and Password) to try to establish a connection to the "real" Director. If that succeeds, all further incoming console commands are directly passed on to that Director.
4. The Console submits the backup Job `run` command with the new option `fdcalled=1`.
5. The Director checks against ACLs configured inside its Console resource representing the current connection if the Job is allowed, and if it is, it first marks the existing connection from the FD as related to this Job, and then runs the Job.
6. When the DIR starts using the established connection for a Job, the FD closes the connection to the incoming console; the connection to the DIR is used exclusively for Job-related communication, and thus a console connected to it could not be used for anything useful from this moment on.
7. The Director, following its regular mode of operation, eventually sends a command to the FD telling it which Storage Daemon to contact for data transfers.

8. The File Daemon then connects to the SD as usual.
9. The Backup Job progresses as usual, and at the end, the FD will report the final Job status to the DIR.
10. Eventually, the TCP connection between FD and DIR will be properly shutdown.

The above step-by-step description already indicates where specific configuration for this operational mode is required:

- The console program (either `bconsole` or the `Tray Monitor`) needs to be configured to connect to a File Daemon (usually the one running locally) instead of directly to a DIR.
- The File Daemon used needs to be configured with at least one additional Director resource with specific settings, one of them being the reference to a `Console` resource.
- In the Director's configuration, a client-specifc `Console` resource should be used, and this should be restricted in capabilities as far as possible.
- A Job that has to use a FD-initiated connection needs to be started with a specific new option, namely `fdcalled=1`.

For an experienced Bacula administrator, having to add `Console` and more `Director` resources to a File Daemon configuration will be unexpected. For new Bacula administrators, the complexity of the configuration scheme (which, as usual, is a result of the built-in flexibility) may be intimidating.

For this reason, we provide an overview of the required configuration in figure 49.4 on the facing page.

In the overview figure, we have included a Bacula **Console** configuration which has not yet been mentioned. However, it works essentially identical to the Tray Monitor program connectivity, but allows – and requires! – the user to run arbitrary commands with all possible options. At this time, this is the only way to restore data using Client-Initiated connections.

The figure 49.4 on the next page shows all needed configuration entities, in some cases excerpts from the full configuration, in others the complete files.

Colored arrows indicate the relationships between resources. Most of them are also relationships between daemons potentially running on separate hosts, accordingly, any of those will require TCP/IP connectivity, including routing, name resolution, firewall and TCP Wrapper configuration. However, managing those aspects should not be an additional effort for Bacula administrators, as no additional requirements are introduced, nor will it be a problem for most users, as the common desktop configurations allow outgoing connections already.

Resource relationships

When looking at the configuration overview in figure 49.4 on the facing page, it is easy to spot distinct relationships. We will discuss them, starting at the console program used.

Console programs intended to use a proxied connection to the Director, or intended to start a Job through a client-initiated connection, need to be configured to contact the File Daemon used to serve as proxy to the Director, not the Director itself. Thus, the configuration will (usually) use **Address = localhost** and a port number of 9102.

Note that `bconsole` is essentially agnostic of the fact that it connects to a File Daemon, not a Director, and thus uses a Director configuration resource, while the Tray Monitor program does know about those features and accordingly uses a slightly different configuration scheme with a Client resource with **Remote = Yes** set.

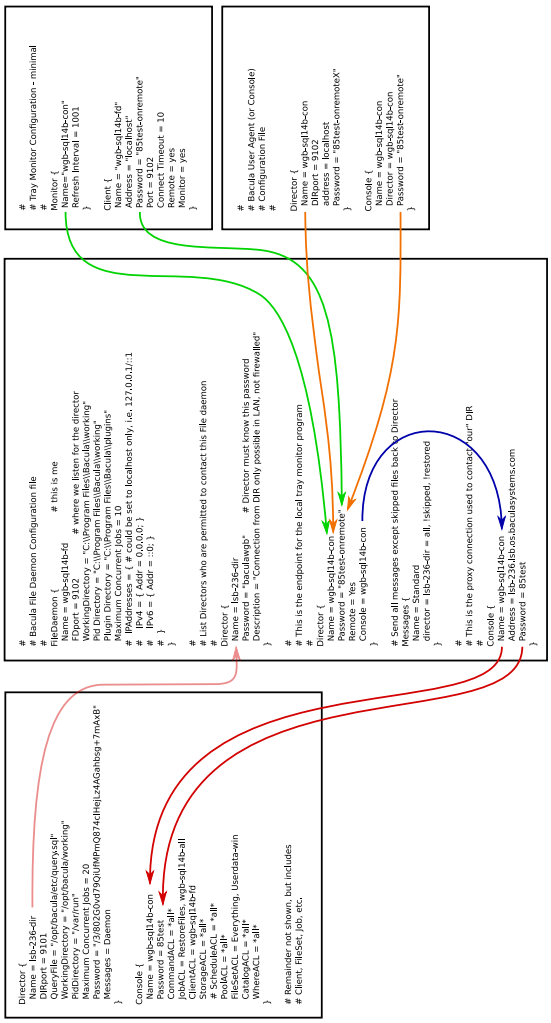


Figure 49.4: Configuration overview: resources and their relationships

Also, bconsole and the Tray Monitor program use different resources to define themselves (which is where the name used for authentication is taken from), so there will be different "Identity" resources in use. It is possible and may be reasonable to use the same name and password, though, as the File Daemon can use the same resource to configure the properties of the connecting agent.

In figure 49.4 on the previous page, the relationship between Tray Monitor and (local) File Daemon configuration resources is shown in green, and the relationship from a bconsole configuration to that of the File Daemon is shown in orange.

File Daemon configuration for proxied connections to a Director are a bit more complex.

First, the File Daemon needs to accept incoming user agent connections from bconsole or the Tray Monitor program. However, these connections will (potentially) not only be used to query the FD's status, but also to submit more powerful commands, so a Monitor resource is not suitable.

Instead, the File Daemon has essentially to act as a Director for the user agent, so this functionality is configured in a Director resource.

This is essentially just a "normal" Director resource, but it is advisable to have distinct ones for "real" Bacula Directors as needed (usually there will only be one "real" Director) and for incoming user agents (also, there will usually be exactly one of them).

This is because direct Director connections may still be required in some cases, and it is surely advisable to have different passwords for a Director and for a user agent. Also, it is much easier to understand log files if different names are used for real Directors and user agent connections.

To enable the proxy functionality of the File Daemon, the directive **Remote** needs to be set to yes for those Director resources representing incoming user agent connections.

In addition, a **Console** directive should be present and indicate a Console resource used to contact the "real" Director – see below for details.

If no Console is referenced, the File Daemon will automatically select a Console it finds in its configuration, but since it is possible to have several of them defined, each with different properties, it is much safer and more clearly understandable what is configured if an explicit Console reference is used.

In the overview in figure 49.4 on the preceding page, this Console reference is indicated in blue.

The Console resource of the File Daemon specifies connection information and credentials to contact a "real" Director as usual.

It is advisable to use distinct names and passwords for each such Director connection, so that access can be controlled at least with the granularity of client machines.

However, if different users on a given machine will be using proxied connections to one or more Directors, there should be distinct Director resources per user, each using its own set of credentials.

As the File Daemon configuration file should be protected against user access, this allows separating user permissions and will allow to distinguish activity logged.

However, the local machine administrator will always be able to read all user credentials, thus needs to be trustworthy to at least the extent all the users combined are trusted.

Additional considerations apply to logging of File Daemon activity. Usually all File Daemon activity is logged to a default Director (Job-related activity is always logged to the initiating Director). However, in environments making use of the Client-Initiated Connection facilities,

the central Director may not be a suitable target for the File Daemon messages as it will be inaccessible most of the time.

Thus we recommend to configure the File Daemon to write its messages to a local log file or logging daemon.

Director configuration for use in an environment using Client-Initiated Connections relies heavily on the use of Named Consoles with ACLs. In the configuration overview of figure 49.4 on page 549, we show one such Console resource and red arrows from the File Daemon's `Console` resource indicate how it is referenced from the (proxying) File Daemon.

We also show the "regular" configuration, with the File Daemon being contacted by the Director (the Director-side Client resource is not in the overview) with a faint red arrow. The important part here is that authentication information is not shared between the two different connection direction setups.

Access Control Lists should be in place in the Director-side configuration for all Console resources that are used by non-administrative users. Only Bacula instance administrators should ever have full access to a Director.

Using the configuration scheme outlined, it is easily possible to have per-user Console configurations, typically restricting access to only the required resources. Most of the time, for example, there will be only one client and one backup Job allowed for such a Console connection.

In the overview graph we present a simplified, not very closely restricted Console resource.

In practice, much more restrictive configurations will be applied. We trust that a Bacula administrator will know which commands their users are supposed to execute, and which resources they need to access. However, with the Tray Monitor program, it may not be obvious which commands in particular are required, thus we provide the full list of mandatory commands here. Note that resource (for example, Job, Pool and Storage) restrictions need to be added.

A typical **Command ACL** directive for a Console used by a Tray Monitor which is allowed to run Jobs will look require to have at least the following contents:

```
CommandACL = status, .estimate, .clients, .jobs, .pools, .storage, .filesets, run
```

Using `bconsole` for User-Initiated Jobs

Besides using the Tray Monitor utility to monitor and control a Director, it is also possible to use the console program `bconsole` with the File Daemon as a proxy, including the ability to start jobs which can use the File Daemon-initiated Director connection.

At the current time, this is actually the only reliable way to enable users with a restricted console connection and without full access from the Director to their desktop to restore from their backups.

To use these facilities, three items are important:

1. The programs involved need to be configured accordingly, i. e. as outlined above.
2. To initiate the proxied connection through the File Daemon, before any command intended for the Director is entered, the command `proxy` needs to be submitted. It should return with an "Ok" message.
3. Any job started that should use the established console connection to the Director needs the additional parameter `fdcalled=1` in its command line.

Note that the extra command and parameter are not required when submitting commands through the Tray Monitor program's command file facility. Also note that once a job has started, the console connection between File Daemon and bconsole will be terminated, causing bconsole to end. Job status can be observed with a new bconsole invocation; in fact, the status of the local File Daemon can then be observed without going through the Director. An example session is shown in figure 49.5.

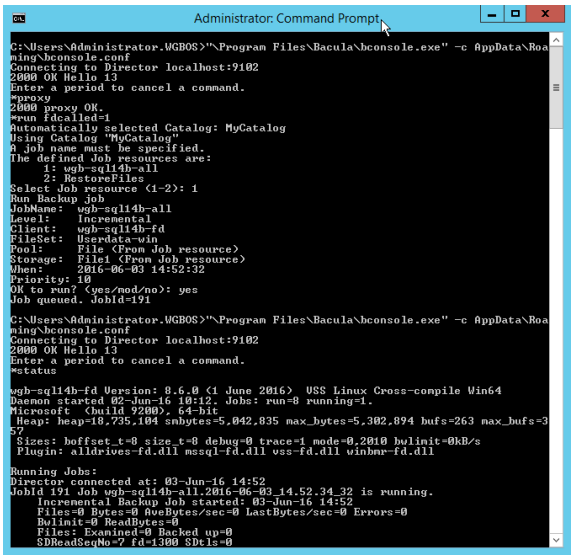


Figure 49.5: Remote, Proxied bconsole Session with Job Run

49.1.7 Monitoring

49.1.8 Running Jobs

49.1.9 Proxied Connection

49.1.10 Supported Platforms

The Tray Monitor program is available for the following platforms:

- Microsoft Windows, both 32 and 64 bit, on all Windows versions still under vendor support.
- Linux:
  - Ubuntu: 14.04, x86 and x86-64, with XFCE desktop
  - Red Hat Enterprise Linux and clones like CentOS: 6.x, 7.x, x86 and x86-64, with XFCE desktop

49.1.11 Command Line Options

The Tray Monitor program accepts the following command line options and parameters:

-c <filename> Configuration file to use. If the file does not exist, the program starts with an empty configuration and shows the configuration panel.

-d <number> Debug level. The higher the number, the more information is displayed.

-dt Print timestamps in debug output.

-t Test the configuration and exit. No output is a good sign.

-W <0|1> Force detection of systray capability. A zero indicates to run without tray icon, a one forces to use tray capability even if not detected.

-? Display short help and exit.

Chapter 50

Installing and Configuring MySQL

50.1 Installing and Configuring MySQL – Phase I

If you use the `./configure --with-mysql=mysql-directory` statement for configuring **Bacula**, you will need MySQL version 4.1 or later installed in the `mysql-directory`. If you are using one of the new modes such as ANSI/ISO compatibility, you may experience problems.

If MySQL is installed in the standard system location, you need only enter --**with-mysql** since the configure program will search all the standard locations. If you install MySQL in your home directory or some other non-standard directory, you will need to provide the full path to it.

Installing and Configuring MySQL is not difficult but can be confusing the first time. As a consequence, below, we list the steps that we used to install it on our machines. Please note that our configuration leaves MySQL without any user passwords. This may be an undesirable situation if you have other users on your system.

The notes below describe how to build MySQL from the source tar files. If you have a pre-installed MySQL, you can return to complete the installation of Bacula, then come back to Phase II of the MySQL installation. If you wish to install MySQL from rpms, you will probably need to install the following:

```
mysql-<version>.rpm
mysql-server-<version>.rpm
mysql-devel-<version>.rpm
```

If you wish to install them from debs, you will probably need the following:

```
mysql-server-<version>.deb
mysql-client-<version>.deb
libmysqlclient15-dev-<version>.deb
libmysqlclient15off-<version>.deb
```

The names of the packages may vary from distribution to distribution. It is important to have the **devel** or **dev** package loaded as it contains the libraries and header files necessary to build Bacula. There may be additional packages that are required to install the above, for example, `zlib` and `openssl`.

Once these packages are installed, you will be able to build Bacula (using the files installed with the mysql package, then run MySQL using the files installed with mysql-server. If you have installed MySQL by debs or rpms, please skip Phase I below, and return to complete the installation of Bacula, then come back to Phase II of the MySQL installation when indicated to do so.

Beginning with Bacula version 1.31, the thread safe version of the MySQL client library is used, and hence you should add the --**enable-thread-safe-client** option to the ./configure as shown below:

1. Download MySQL source code from www.mysql.com/downloads
2. Detar it with something like:

   ```
   tar xvfz mysql-filename
   ```

   Note, the above command requires GNU tar. If you do not have GNU tar, a command such as:

   ```
   zcat mysql-filename | tar xvf -
   ```

   will probably accomplish the same thing.
3. ```
   cd mysql-source-directory
   ```

   where you replace mysql-source-directory with the directory name where you put the MySQL source code.
4. ```
   ./configure --enable-thread-safe-client --prefix=mysql-directory
   ```

   where you replace mysql-directory with the directory name where you want to install mysql. Normally for system wide use this is /usr/local/mysql. In my case, I use ~kern/mysql.
5. ```
   make
   ```

   This takes a bit of time.
6. make install

   This will put all the necessary binaries, libraries and support files into the mysql-directory that you specified above.
7. ```
   ./scripts/mysql\_install\_db
   ```

   This will create the necessary MySQL databases for controlling user access. Note, this script can also be found in the bin directory in the installation directory

The MySQL client library **mysqlclient** requires the gzip compression library libz.a or libz.so. If you are using rpm packages, these libraries are in the **libz-devel** package. On Debian systems, you will need to load the **zlib1g-dev** package. If you are not using rpms or debs, you will need to find the appropriate package for your system.

At this point, you should return to completing the installation of **Bacula**. Later after Bacula is installed, come back to this chapter to complete the installation. Please note, the installation files used in the second phase of the MySQL installation are created during the Bacula Installation.

50.2 Installing and Configuring MySQL – Phase II

At this point, you should have built and installed MySQL, or already have a running MySQL, and you should have configured, built and installed **Bacula**. If not, please complete these items before proceeding.

Please note that the ./configure used to build **Bacula** will need to include --**with-mysql=mysql-directory**, where mysql-directory is the directory name that you specified on the ./configure command for configuring MySQL. This is needed so that Bacula can find the necessary include headers and library files for interfacing to MySQL.

Bacula will install scripts for manipulating the database (create, delete, make tables etc) into the main installation directory. These files will be of the form *_bacula_* (e.g. create_bacula_database). These files are also available in the <bacula-src>/src/cats directory after running ./configure. If you inspect create_bacula_database, you will see that it calls create_mysql_database. The *_bacula_* files are provided for convenience. It doesn't matter what database you have chosen; create_bacula_database will always create your database.

Now you will create the Bacula MySQL database and the tables that Bacula uses.

1. Start mysql. You might want to use the startmysql script provided in the Bacula release.
2. ```
   cd <install-directory>
   ```
   This directory contains the Bacula catalog interface routines.
3. ```
   ./grant_mysql_privileges
   ```
   This script creates unrestricted access rights for the user **bacula**. You may want to modify it to suit your situation. Please note that none of the userids, including root, are password protected. If you need more security, please assign a password to the root user and to bacula. The program mysqladmin can be used for this.
4. ```
   ./create_mysql_database
   ```
   This script creates the MySQL **bacula** database. The databases you create as well as the access databases will be located in <install-dir>/var/ in a subdirectory with the name of the database, where <install-dir> is the directory name that you specified on the **prefix** option. This can be important to know if you want to make a special backup of the Bacula database or to check its size.
5. ```
   ./make_mysql_tables
   ```
   This script creates the MySQL tables used by **Bacula**.

Each of the three scripts (grant_mysql_privileges, create_mysql_database and make_mysql_tables) allows the addition of a command line argument. This can be useful for specifying the user and or password. For example, you might need to add **-u root** to the command line to have sufficient privilege to create the Bacula tables.

To take a closer look at the access privileges that you have setup with the above, you can do:

```
<mysql-directory>/bin/mysql -u root mysql
select * from user;
```

50.3 Re-initializing the Catalog Database

After you have done some initial testing with **Bacula**, you will probably want to re-initialize the catalog database and throw away all the test Jobs that you ran. To do so, you can do the following:

```
cd <install-directory>
./drop_mysql_tables
./make_mysql_tables
```

Please note that all information in the database will be lost and you will be starting from scratch. If you have written on any Volumes, you must write an end of file mark on the volume so that Bacula can reuse it. Do so with:

```
(stop Bacula or unmount the drive)
mt -f /dev/nst0 rewind
mt -f /dev/nst0 weof
```

Where you should replace /dev/nst0 with the appropriate tape drive device name for your machine.

50.4 Linking Bacula with MySQL

After configuring Bacula with

./configure --enable-thread-safe-client -prefix=<mysql-directory> where <mysql-directory> is in my case /home/kern/mysql, you may have to configure the loader so that it can find the MySQL shared libraries. If you have previously followed this procedure and later add the **--enable-thread-safe-client** options, you will need to rerun the ldconfig program shown below. If you put MySQL in a standard place such as /usr/lib or /usr/local/lib this will not be necessary, but in my case it is. The description that follows is Linux specific. For other operating systems, please consult your manuals on how to do the same thing:

First edit: /etc/ld.so.conf and add a new line to the end of the file with the name of the mysql-directory. In my case, it is:

/home/kern/mysql/lib/mysql then rebuild the loader's cache with:

/sbin/ldconfig. If you upgrade to a new version of **MySQL**, the shared library names will probably change, and you must re-run the /sbin/ldconfig command so that the runtime loader can find them.

Alternatively, your system my have a loader environment variable that can be set. For example, on a Solaris system where I do not have root permission, I use:

```
LD_LIBRARY_PATH=/home/kern/mysql/lib/mysql
```

Finally, if you have encryption enabled in MySQL, you may need to add **-lssl -lcrypto** to the link. In that case, you can either export the appropriate LDFLAGS definition, or alternatively, you can include them directly on the ./configure line as in:

```
LDFLAGS="-lssl -lcyrpto" ./configure <your-options>
```

50.5 Installing MySQL from RPMs

If you are installing MySQL from RPMs, you will need to install both the MySQL binaries and the client libraries. The client libraries are usually found in a devel package, so you must install:

```
mysql
mysql-devel
```

This will be the same with most other package managers too.

50.6 Upgrading MySQL

If you upgrade MySQL, you must reconfigure, rebuild, and re-install Bacula otherwise you are likely to get bizarre failures. If you install from rpms and you upgrade MySQL, you must also

rebuild Bacula. You can do so by rebuilding from the source rpm. To do so, you may need to modify the bacula.spec file to account for the new MySQL version.

50.7 MySQL Configuration Caution

Bacula requires that at least one of the **TIMESTAMP** fields in the database schema to have a valid value of zero. If you turn on the **SQL_MODE STRICT_ALL_TABLES**, you must be sure that you turn off the **NO_ZERO_DATA** and the **NO_ZERO_IN_DATE** restrictions or Bacula will not function correctly.

Chapter 51

Installing and Configuring PostgreSQL

If you are considering using PostgreSQL, you should be aware of their philosophy of upgrades, which could be destabilizing for a production shop. Basically at every major version upgrade, you are required to dump your database in an ASCII format, do the upgrade, and then reload your database (or databases). This is because they frequently update the "data format" from version to version, and they supply no tools to automatically do the conversion. If you forget to do the ASCII dump, your database may become totally useless because none of the new tools can access it due to the format change, and the PostgreSQL server will not be able to start.

If you are building PostgreSQL from source, please be sure to add the **--enable-thread-safety** option when doing the `./configure` for PostgreSQL.

51.1 Installing PostgreSQL

If you use the `./configure --with-postgresql=PostgreSQL-Directory` statement for configuring **Bacula**, you will need PostgreSQL version 7.4 or later installed. NOTE! PostgreSQL versions earlier than 7.4 do not work with Bacula. If PostgreSQL is installed in the standard system location, you need only enter **--with-postgresql** since the configure program will search all the standard locations. If you install PostgreSQL in your home directory or some other non-standard directory, you will need to provide the full path with the **--with-postgresql** option.

Installing and configuring PostgreSQL is not difficult but can be confusing the first time. If you prefer, you may want to use a package provided by your chosen operating system. Binary packages are available on most PostgreSQL mirrors.

If you prefer to install from source, we recommend following the instructions found in the PostgreSQL documentation[1].

If you are using FreeBSD, this FreeBSD Diary article[2] will be useful. Even if you are not using FreeBSD, the article will contain useful configuration and setup information.

If you configure the Batch Insert code in Bacula (attribute inserts are 10 times faster), you **must** be using a PostgreSQL that was built with the **--enable-thread-safety** option, otherwise you will get data corruption. Most major Linux distros have thread safety turned on, but it is better to check. One way is to see if the PostgreSQL library that Bacula will be linked against references pthreads. This can be done with a command such as:

[1] http://www.postgresql.org/docs/
[2] http://www.freebsddiary.org/postgresql.php

```
nm /usr/lib/libpq.a | grep pthread_mutex_lock
```

The above command should print a line that looks like:

```
U pthread_mutex_lock
```

if does, then everything is OK. If it prints nothing, do not enable batch inserts when building Bacula.

After installing PostgreSQL, you should return to completing the installation of **Bacula**. Later, after Bacula is installed, come back to this chapter to complete the installation. Please note, the installation files used in the second phase of the PostgreSQL installation are created during the Bacula Installation. You must still come back to complete the second phase of the PostgreSQL installation even if you installed binaries (e.g. rpm, deb, ...).

51.2 Configuring PostgreSQL

At this point, you should have built and installed PostgreSQL, or already have a running PostgreSQL, and you should have configured, built and installed **Bacula**. If not, please complete these items before proceeding.

Please note that the ./configure used to build **Bacula** will need to include **--with-postgresql=PostgreSQL-directory**, where **PostgreSQL-directory** is the directory name that you specified on the ./configure command for configuring PostgreSQL (if you didn't specify a directory or PostgreSQL is installed in a default location, you do not need to specify the directory). This is needed so that Bacula can find the necessary include headers and library files for interfacing to PostgreSQL.

An important thing to note here is that **Bacula** makes two connections to the PostgreSQL server for each backup job that is currently running. If you are intending to run a large number of concurrent jobs, check the value of **max_connections** in your PostgreSQL configuration file to ensure that it is larger than the setting **Maximum Concurrent Jobs** in your director configuration. **Setting this too low will result in some backup jobs failing to run correctly!**

Bacula will install scripts for manipulating the database (create, delete, make tables etc) into the main installation directory. These files will be of the form *_bacula_* (e.g. create_bacula_database). These files are also available in the <bacula-src>/src/cats directory after running ./configure. If you inspect create_bacula_database, you will see that it calls create_postgresql_database. The *_bacula_* files are provided for convenience. It doesn't matter what database you have chosen; create_bacula_database will always create your database.

Now you will create the Bacula PostgreSQL database and the tables that Bacula uses. These instructions assume that you already have PostgreSQL running. You will need to perform these steps as a user that is able to create new databases. This can be the PostgreSQL user (on most systems, this is the pgsql user).

1. cd <install-directory>

   This directory contains the Bacula catalog interface routines.

2. Create the database owner (**bacula**) On many systems, the PostgreSQL master owner is pgsql and on others such as Red Hat and Fedora it is postgres. You can find out which it is by examining your /etc/passwd file. To create a new user under either your name or with say the name bacula, you can do the following:

   ```
   su
   (enter root password)
   ```

```
su pgsql (for rpm systems or postgres for Debian systems)
createuser bacula
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) (choose
      what you want)
exit
```

If you have a newer PostgreSQL, you may need to use the **-s** option on the createuser command. Example:

```
createuser -s bacula
```

Normally the bacula user must be able to create new databases, if you use the script in the next item, or you will have to create one for it, but it does not need to create new users.

3. ./create_bacula_database

   This script creates the PostgreSQL **bacula** database. Before running this command, you should carefully think about what encoding sequence you want for the text fields (paths, files, ...). We strongly recommend that you use the default value of SQL_ASCII that is in the create_bacula_database script. Please be warned that if you change this value, your backups may fail. After running the script, you can check with the command:

   ```
   psql -l
   ```

   and the column marked **Encoding** should be **SQL_ASCII** for all your Bacula databases (normally **bacula**).

4. ./make_bacula_tables

   This script creates the PostgreSQL tables used by **Bacula**.

5. ./grant_bacula_privileges

   This script creates the database user **bacula** with restricted access rights. You may want to modify it to suit your situation. Please note that this database is not password protected.

Each of the three scripts (create_bacula_database, make_bacula_tables, and grant_bacula_privileges) allows the addition of a command line argument. This can be useful for specifying the user name. For example, you might need to add **-h hostname** to the command line to specify a remote database server.

To take a closer look at the access privileges that you have setup with the above, you can do:

```
PostgreSQL-directory/bin/psql --command \dp bacula
```

For people who are not Postgresql experts, it is sometimes difficult to get the authorization working correctly with Bacula. One simple, but not recommended way, to authorize Bacula is to modify your **pg_hba.conf** file (in /var/lib/pgsql/data or in /var/lib/postgresql/8.x or in /etc/postgres/8.x/main on other distributions) from:

```
local   all    all        ident  sameuser
```

to

```
local   all    all        trust
```

This is a quick way to solve the problem, but it is not always a good thing to do from a security standpoint. However, it allows one to run my regression scripts without having a password.

A more secure way to perform database authentication is with md5 password hashes. Begin by editing the **pg_hba.conf** file, and above the existing **local** and **host** lines, add the line:

```
local bacula bacula md5
```

then restart the PostgreSQL database server (frequently, this can be done using /etc/init.d/postgresql restart or service postgresql restart) to put this new authentication rule into effect.

More detailed information on the **pg_hba.conf** file can be found at PostgreSQL pg_hba.conf Documentation

Next, become the Postgres administrator, postgres, either by logging on as the postgres user, or by using su to become root and then using **su - postgres** or **su - pgsql** to become postgres. Add a password to the **bacula** database for the **bacula** user using:

```
$ psql bacula
bacula=# alter user bacula with password 'secret';
ALTER USER
bacula=# \q
```

You'll have to add this password to two locations in the bacula-dir.conf file: once to the Catalog resource and once to the RunBeforeJob entry in the BackupCatalog Job resource. With the password in place, these two lines should look something like:

```
dbname = bacula; user = bacula; password = "secret"
```

. . . and . . .

```
# WARNING!!! Passing the password via the command line is insecure.
# see comments in make_catalog_backup.pl for details.
RunBeforeJob = "/etc/make_catalog_backup.pl MyCatalog"
```

Naturally, you should choose your own significantly more random password, and ensure that the bacula-dir.conf file containing this password is readable only by the root user.

Even with the files containing the database password properly restricted, there is still a security problem with this approach: on some platforms, the environment variable that is used to supply the password to PostgreSQL is available to all users of the local system. To eliminate this problem, the PostgreSQL team have deprecated the use of the environment variable password-passing mechanism and recommend the use of a .pgpass file instead. To use this mechanism, create a file named .pgpass containing the single line:

```
localhost:5432:bacula:bacula:secret
```

This file should be copied into the home directory of all accounts that will need to gain access to the database: typically, root, bacula, and any users who will make use of any of the console programs. The files must then have the owner and group set to match the user (so root:root for the copy in root, and so on), and the mode set to 600, limiting access to the owner of the file.

51.3 Re-initializing the Catalog Database

After you have done some initial testing with **Bacula**, you will probably want to re-initialize the catalog database and throw away all the test Jobs that you ran. To do so, you can do the following:

```
cd <install-directory>
./drop_bacula_tables
./make_bacula_tables
./grant_bacula_privileges
```

Please note that all information in the database will be lost and you will be starting from scratch. If you have written on any Volumes, you must write an end of file mark on the volume so that Bacula can reuse it. Do so with:

```
(stop Bacula or unmount the drive)
mt -f /dev/nst0 rewind
mt -f /dev/nst0 weof
```

Where you should replace /dev/nst0 with the appropriate tape drive device name for your machine.

51.4 Installing PostgreSQL from RPMs

If you are installing PostgreSQL from RPMs, you will need to install both the PostgreSQL binaries and the client libraries. The client libraries are usually found in a **devel** or **dev** package, so you must install the following for rpms:

```
postgresql
postgresql-devel
postgresql-server
postgresql-libs
```

and the following for debs:

```
postgresql
postgresql-common
postgresql-client
postgresql-client-common
libpq5
libpq-dev
```

These will be similar with most other package managers too. After installing from rpms, you will still need to run the scripts that set up the database and create the tables as described above.

51.5 Converting from MySQL to PostgreSQL

There are various scripts available that allow you to convert from MySQL to PostgreSQL database format. The first step is to backup your existing MySQL database, then shutdown Bacula.

One such conversion program from MySQL to PostgreSQL can be found at: MySQL to PostgreSQL Conversion Scripts

51.6 Upgrading PostgreSQL

Bacula Enterprise is built with the PostgreSQL that is appropriate for each Operating System platform. Each time that PostgreSQL is updated, the Bacula binaries must be updated. If you upgrade PostgreSQL, on older PostgreSQLs, you needed to rebuild Bacula for the specific version of PostgreSQL. However, on more recent versions, this is no longer necessary. If you experience problems after upgrading PostgreSQL, you might try rebuilding Bacula.

51.7 Tuning PostgreSQL

If you despool attributes for many jobs at the same time, you can tune the sequence object for the FileId field.

```
psql -Ubacula bacula

ALTER SEQUENCE file_fileid_seq CACHE 1000;
```

51.8 Credits

Many thanks to Dan Langille for writing the PostgreSQL driver. This will surely become the most popular database that Bacula supports.

Chapter 52

Catalog Maintenance

Without proper setup and maintenance, your Catalog may continue to grow indefinitely as you run Jobs and backup Files, and/or it may become very inefficient and slow. How fast the size of your Catalog grows depends on the number of Jobs you run and how many files they backup. By deleting records within the database, you can make space available for the new records that will be added during the next Job. By constantly deleting old expired records (dates older than the Retention period), your database size will remain constant.

If you started with the default configuration files, they already contain reasonable defaults for a small number of machines (less than 5), so if you fall into that case, catalog maintenance will not be urgent if you have a few hundred megabytes of disk space free. Whatever the case may be, some knowledge of retention periods will be useful.

52.1 Setting Retention Periods

Bacula uses three Retention periods: the **File Retention** period, the **Job Retention** period, and the **Volume Retention** period. Of these three, the File Retention period is by far the most important in determining how large your database will become.

The **File Retention** and the **Job Retention** are specified in each Client resource as is shown below. The **Volume Retention** period is specified in the Pool resource, and the details are given in the next chapter of this manual.

File Retention = <time-period-specification> The File Retention record defines the length of time that Bacula will keep File records in the Catalog database. When this time period expires, and if **AutoPrune** is set to **yes**, Bacula will prune (remove) File records that are older than the specified File Retention period. The pruning will occur at the end of a backup Job for the given Client. Note that the Client database record contains a copy of the File and Job retention periods, but Bacula uses the current values found in the Director's Client resource to do the pruning.

Since File records in the database account for probably 80 percent of the size of the database, you should carefully determine exactly what File Retention period you need. Once the File records have been removed from the database, you will no longer be able to restore individual files in a Job. However, with Bacula version 1.37 and later, as long as the Job record still exists, you will be able to restore all files in the job.

Retention periods are specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years on the record. See the Configuration chapter of this manual for additional details of modifier specification.

The default File retention period is 60 days.

Job Retention = <time-period-specification> The Job Retention record defines the length of time that **Bacula** will keep Job records in the Catalog database. When this time period expires, and if **AutoPrune** is set to **yes** Bacula will prune (remove) Job records that are older than the specified Job Retention period. Note, if a Job record is selected for pruning, all associated File and JobMedia records will also be pruned regardless of the File Retention period set. As a consequence, you normally will set the File retention period to be less than the Job retention period.

As mentioned above, once the File records are removed from the database, you will no longer be able to restore individual files from the Job. However, as long as the Job record remains in the database, you will be able to restore all the files backuped for the Job (on version 1.37 and later). As a consequence, it is generally a good idea to retain the Job records much longer than the File records.

The retention period is specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of modifier specification.

The default Job Retention period is 180 days.

AutoPrune = <yes|no> If AutoPrune is set to yes (default), Bacula will automatically apply the File retention period and the Job retention period for the Client at the end of the Job.

If you turn this off by setting it to **no**, your Catalog will grow each time you run a Job.

52.2 Compacting Your MySQL Database

Over time, as noted above, your database will tend to grow. I've noticed that even though Bacula regularly prunes files, MySQL does not effectively use the space, and instead continues growing. To avoid this, from time to time, you must compact your database. Normally, large commercial database such as Oracle have commands that will compact a database to reclaim wasted file space. MySQL has the **OPTIMIZE TABLE** command that you can use. We leave it to you to explore the utility of the **OPTIMIZE TABLE** command in MySQL.

All database programs have some means of writing the database out in ASCII format and then reloading it. Doing so will re-create the database from scratch producing a compacted result, so below, we show you how you can do this for MySQL, PostgreSQL.

For a **MySQL** database, you could write the Bacula database as an ASCII file (bacula.sql) then reload it by doing the following:

```
mysqldump -f --opt bacula > bacula.sql
mysql bacula < bacula.sql
rm -f bacula.sql
```

Depending on the size of your database, this will take more or less time and a fair amount of disk space. For example, if I cd to the location of the MySQL Bacula database (typically /opt/mysql/var or something similar) and enter:

```
du bacula
```

I get **620,644** which means there are that many blocks containing 1024 bytes each or approximately 635 MB of data. After doing the mysqldump, I had a bacula.sql file that had **174,356** blocks, and after doing the mysql command to recreate the database, I ended up with a total of **210,464** blocks rather than the original **629,644**. In other words, the compressed version of the database took approximately one third of the space of the database that had been in use for about a year.

As a consequence, I suggest you monitor the size of your database and from time to time (once every six months or year), compress it.

52.3 Repairing Your MySQL Database

If you find that you are getting errors writing to your MySQL database, or Bacula hangs each time it tries to access the database, you should consider running MySQL's database check and repair routines. The program you need to run depends on the type of database indexing you are using. If you are using the default, you will probably want to use myisamchk. For more details on how to do this, please consult the MySQL document at: www.mysql.com/doc/en/Repair.html.

If the errors you are getting are simply SQL warnings, then you might try running dbcheck before (or possibly after) using the MySQL database repair program. It can clean up many of the orphaned record problems, and certain other inconsistencies in the Bacula database.

A typical cause of MySQL database problems is if your partition fills. In such a case, you will need to create additional space on the partition or free up some space then repair the database probably using myisamchk. Recently my root partition filled and the MySQL database was corrupted. Simply running myisamchk -r did not fix the problem. However, the following script did the trick for me:

```
#!/bin/sh
for i in *.MYD ; do
  mv $i x${i}
  t=`echo $i | cut -f 1 -d '.' -`
  mysql bacula <<END_OF_DATA
set autocommit=1;
truncate table $t;
quit
END_OF_DATA
  cp x${i} ${i}
  chown mysql:mysql ${i}
  myisamchk -r ${t}
done
```

I invoked it with the following commands:

```
cd /var/lib/mysql/bacula
./repair
```

Then after ensuring that the database was correctly fixed, I did:

```
cd /var/lib/mysql/bacula
rm -f x*.MYD
```

52.4 MySQL Table is Full

If you are running into the error **The table 'File' is full ...**, it is probably because on version 4.x MySQL, the table is limited by default to a maximum size of 4 GB and you have probably run into the limit. The solution can be found at: dev.mysql.com/doc/refman/5.0/en/full-table.html

You can display the maximum length of your table with:

```
mysql bacula
SHOW TABLE STATUS FROM bacula like "File";
```

If the column labeled "Max_data_length" is around 4Gb, this is likely to be the source of your problem, and you can modify it with:

```
mysql bacula
ALTER TABLE File MAX_ROWS=281474976710656;
```

Alternatively you can modify your /etc/my.conf file before creating the Bacula tables, and in the [mysqld] section set:

```
set-variable = myisam_data_pointer_size=6
```

The above myisam data pointer size must be made before you create your Bacula tables or it will have no effect.

The row and pointer size changes should already be the default on MySQL version 5.x, so making these changes should only be necessary on MySQL 4.x depending on the size of your catalog database.

52.5 MySQL Server Has Gone Away

If you are having problems with the MySQL server disconnecting or with messages saying that your MySQL server has gone away, then please read the MySQL documentation, which can be found at:

dev.mysql.com/doc/refman/5.0/en/gone-away.html

52.6 MySQL Temporary Tables

When doing backups with large numbers of files, MySQL creates some temporary tables. When these tables are small they can be held in system memory, but as they approach some size, they spool off to disk. The default location for these temp tables is /tmp. Once that space fills up, Bacula daemons such as the Storage daemon doing spooling can get strange errors. E.g.

```
Fatal error: spool.c:402 Spool data read error.
Fatal error: backup.c:892 Network send error to SD. ERR=Connection reset by peer
```

What you need to do is setup MySQL to use a different (larger) temp directory, which can be set in the /etc/my.cnf with these variables set:

```
tmpdir=/path/to/larger/tmpdir
bdb_tmpdir=/path/to/larger/tmpdir
```

52.7 Repairing Your PostgreSQL Database

The same considerations apply that are indicated above for MySQL. That is, consult the PostgreSQL documents for how to repair the database, and also consider using Bacula's dbcheck program if the conditions are reasonable for using (see above).

52.8 Database Performance Issues

There are a considerable number of ways each of the databases can be tuned to improve the performance. Going from an untuned database to one that is properly tuned can make a difference of a factor of 100 or more in the time to insert or search for records.

For each of the databases, you may get significant improvements by adding additional indexes. The comments in the Bacula make_xxx_tables give some indications as to what indexes may be appropriate. Please see below for specific instructions on checking indexes.

For MySQL, what is very important is to use the examine the my.cnf file (usually in /etc/my.cnf). You may obtain significant performances by switching to the my-large.cnf or my-huge.cnf files that come with the MySQL source code.

For PostgreSQL, you might want to consider turning fsync off. Of course doing so can cause corrupted databases in the event of a machine crash. There are many different ways that you can tune PostgreSQL, the following document discusses a few of them: www.varlena.com/varlena/GeneralBits/Tidbits/perf.html.

There is also a PostgreSQL FAQ question number 3.3 that may answer some of your questions about how to improve performance of the PostgreSQL engine: www.postgresql.org/docs/faqs.FAQ.html#3.3.

Also for PostgreSQL, look at what "effective_cache_size". For a 2GB memory machine, you probably want to set it at 131072, but don't set it too high. In addition, for a 2GB system, work_mem = 256000 and maintenance_work_mem = 256000 seem to be reasonable values. Make sure your checkpoint_segments is set to at least 8.

52.9 Performance Issues Indexes

One of the most important considerations for improving performance on the Bacula database is to ensure that it has all the appropriate indexes. Several users have reported finding that their database did not have all the indexes in the default configuration. In addition, you may find that because of your own usage patterns, you need additional indexes.

The most important indexes for performance are the two indexes on the **File** table. The first index is on **FileId** and is automatically made because it is the unique key used to access the table. The other one is the (JobId, PathId, Filename) index. If these Indexes are not present, your performance may suffer a lot.

52.9.1 PostgreSQL Indexes

On PostgreSQL, you can check to see if you have the proper indexes using the following commands:

```
psql bacula
select * from pg_indexes where tablename='file';
```

If you do not see output that indicates that all three indexes are created, you can create the two additional indexes using:

```
psql bacula
CREATE INDEX file_jobid_idx on file (jobid);
CREATE INDEX file_jpf_idx on file (jobid, pathid, filename);
```

Make sure that you doesn't have an index on File (filenameid, pathid).

52.9.2 MySQL Indexes

On MySQL, you can check if you have the proper indexes by:

```
mysql bacula
show index from File;
```

If the indexes are not present, especially the JobId index, you can create them with the following commands:

```
mysql bacula
CREATE INDEX file_jobid_idx on File (JobId);
CREATE INDEX file_jpf_idx on File (JobId, PathId, Filename(255));
```

Though normally not a problem, you should ensure that the indexes defined for Filename and Path are both set to 255 characters. Some users reported performance problems when their indexes were set to 50 characters. To check, do:

```
mysql bacula
show index from Path;
```

and what is important is that for Filename, you have an index with Key_name "Name" and Sub_part "255". For Path, you should have a Key_name "Path" and Sub_part "255". If one or the other does not exist or the Sub_part is less that 255, you can drop and recreate the appropriate index with:

```
mysql bacula
DROP INDEX Path on Path;
CREATE INDEX Path on Path (Path(255));
```

52.10 Compacting Your PostgreSQL Database

Over time, as noted above, your database will tend to grow. I've noticed that even though Bacula regularly prunes files, PostgreSQL has a VACUUM command that will compact your database for you. Alternatively you may want to use the vacuumdb command, which can be run from a cron job.

All database programs have some means of writing the database out in ASCII format and then reloading it. Doing so will re-create the database from scratch producing a compacted result, so below, we show you how you can do this for PostgreSQL.

For a **PostgreSQL** database, you could write the Bacula database as an ASCII file (bacula.sql) then reload it by doing the following:

```
pg_dump -c bacula > bacula.sql
cat bacula.sql | psql bacula
rm -f bacula.sql
```

Depending on the size of your database, this will take more or less time and a fair amount of disk space. For example, you can cd to the location of the Bacula database (typically /usr/local/pgsql/data or possible /var/lib/pgsql/data) and check the size.

There are certain PostgreSQL users who do not recommend the above procedure. They have the following to say: PostgreSQL does not need to be dumped/restored to keep the database efficient. A normal process of vacuuming will prevent the database from every getting too large. If you want to fine-tweak the database storage, commands such as VACUUM FULL, REINDEX, and CLUSTER exist specifically to keep you from having to do a dump/restore.

Finally, you might want to look at the PostgreSQL documentation on this subject at www.postgresql.org/docs/8.1/interactive/maintenance.html.

52.11 Migrating from SQLite to MySQL or PostgreSQL

On some older Bacula you may begun using Bacula with SQLite then later find that you want to switch to MySQL or Postgres for any of a number of reasons: SQLite is no longer supported by Bacula; SQLite tends to use more disk than MySQL; when the database is corrupted it is often more catastrophic than with MySQL or PostgreSQL. Several users have succeeded in converting by exporting the SQLite data and then processing it with Perl scripts prior to putting it into MySQL or PostgreSQL. This is, however, not always a simple process. Scripts are available on bacula source distribution under examples/database.

52.12 Backing Up Your Bacula Database

If ever the machine on which your Bacula database crashes, and you need to restore from backup tapes, one of your first priorities will probably be to recover the database. Although Bacula will happily backup your catalog database if it is specified in the FileSet, this is not a very good way to do it, because the database will be saved while Bacula is modifying it. Thus the database may be in an unstable state. Worse yet, you will backup the database before all the Bacula updates have been applied.

To resolve these problems, you need to backup the database after all the backup jobs have been run. In addition, you will want to make a copy while Bacula is not modifying it. To do so, you can use two scripts provided in the release make_catalog_backup.pl and delete_catalog_backup.pl. These files will be automatically generated along with all the other Bacula scripts. The first script will make an ASCII copy of your Bacula database into bacula.sql in the working directory you specified in your configuration, and the second will delete the bacula.sql file.

The basic sequence of events to make this work correctly is as follows:

- Run all your nightly backups
- After running your nightly backups, run a Catalog backup Job
- The Catalog backup job must be scheduled after your last nightly backup
- You use **RunBeforeJob** to create the ASCII backup file and **RunAfterJob** to clean up

Assuming that you start all your nightly backup jobs at 1:05 am (and that they run one after another), you can do the catalog backup with the following additional Director configuration statements:

```
# Backup the catalog database (after the nightly save)
Job {
  Name = "BackupCatalog"
  Type = Backup
  Client=rufus-fd
  FileSet="Catalog"
  Schedule = "WeeklyCycleAfterBackup"
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
  # WARNING!!! Passing the password via the command line is insecure.
  # see comments in make_catalog_backup.pl for details.
  RunBeforeJob = "/opt/bacula/scripts/make_catalog_backup.pl MyCatalog"
  RunAfterJob  = "/opt/bacula/scripts/delete_catalog_backup"
  Write Bootstrap = "/opt/bacula/working/BackupCatalog.bsr"
}
# This schedule does the catalog. It starts after the WeeklyCycle
Schedule {
  Name = "WeeklyCycleAfterBackup
  Run = Level=Full sun-sat at 1:10
}
# This is the backup of the catalog
```

```
FileSet {
  Name = "Catalog"
  Include {
    Options {
      signature=MD5
    }
    File = /opt/bacula/working/bacula.sql
  }
}
```

Be sure to write a bootstrap file as in the above example. However, it is preferable to write or copy the bootstrap file to another computer. It will allow you to quickly recover the database backup should that be necessary. If you do not have a bootstrap file, it is still possible to recover your database backup, but it will be more work and take longer.

52.13 Security considerations

We provide make_catalog_backup.pl as an example of what can be used to backup your Bacula database. We expect you to take security precautions relevant to your situation. make_catalog_backup.pl is designed to take a password on the command line. This is fine on machines with only trusted users. It is not acceptable on machines without trusted users. Most database systems provide a alternative method, which does not place the password on the command line.

The make_catalog_backup.pl script contains some warnings about how to use it. Please read those tips.

To help you get started, we know PostgreSQL has a password file, .pgpass[1], and we know MySQL has .my.cnf[2].

Only you can decide what is appropriate for your situation. We have provided you with a starting point. We hope it helps.

52.14 Backing Up Third Party Databases

If you are running a database in production mode on your machine, Bacula will happily backup the files, but if the database is in use while Bacula is reading it, you may back it up in an unstable state.

The best solution is to shutdown your database before backing it up, or use some tool specific to your database to make a valid live copy perhaps by dumping the database in ASCII format. I am not a database expert, so I cannot provide you advice on how to do this, but if you are unsure about how to backup your database, you might try visiting the Backup Central site, which has been renamed Storage Mountain (www.backupcentral.com). In particular, their Free Backup and Recovery Software page has links to scripts that show you how to shutdown and backup most major databases.

52.15 Database Size

As mentioned above, if you do not do automatic pruning, your Catalog will grow each time you run a Job. Normally, you should decide how long you want File records to be maintained in the Catalog and set the **File Retention** period to that time. Then you can either wait and see

[1] http://www.postgresql.org/docs/8.2/static/libpq-pgpass.html
[2] http://dev.mysql.com/doc/refman/4.1/en/password-security.html

how big your Catalog gets or make a calculation assuming approximately 154 bytes for each File saved and knowing the number of Files that are saved during each backup and the number of Clients you backup.

For example, suppose you do a backup of two systems, each with 100,000 files. Suppose further that you do a Full backup weekly and an Incremental every day, and that the Incremental backup typically saves 4,000 files. The size of your database after a month can roughly be calculated as:

```
Size = 154 * No. Systems * (100,000 * 4 + 10,000 * 26)
```

where we have assumed four weeks in a month and 26 incremental backups per month. This would give the following:

```
Size = 154 * 2 * (100,000 * 4 + 10,000 * 26)
```

or

```
Size = 308 * (400,000 + 260,000)
```

or

```
Size = 203,280,000 bytes
```

So for the above two systems, we should expect to have a database size of approximately 200 Megabytes. Of course, this will vary according to how many files are actually backed up.

Below are some statistics for a MySQL database containing Job records for five Clients beginning September 2001 through May 2002 (8.5 months) and File records for the last 80 days. (Older File records have been pruned). For these systems, only the user files and system files that change are backed up. The core part of the system is assumed to be easily reloaded from the Red Hat rpms.

In the list below, the files (corresponding to Bacula tables) with the extension .MYD contain the data records whereas files with the extension .MYI contain indexes.

You will note that the File records (containing the file attributes) make up the large bulk of the number of records as well as the space used (459 Mega Bytes including the indexes). As a consequence, the most important Retention period will be the **File Retention** period. A quick calculation shows that for each File that is saved, the database grows by approximately 150 bytes.

Size in Bytes	Records	File
168	5	Client.MYD
3,072		Client.MYI
344,394,684	3,080,191	File.MYD
115,280,896		File.MYI
2,590,316	106,902	Filename.MYD
3,026,944		Filename.MYI
184	4	FileSet.MYD
2,048		FileSet.MYI
49,062	1,326	JobMedia.MYD
30,720		JobMedia.MYI
141,752	1,378	Job.MYD
13,312		Job.MYI
1,004	11	Media.MYD
3,072		Media.MYI
1,299,512	22,233	Path.MYD

```
    581,632            Path.MYI
         36          1 Pool.MYD
      3,072            Pool.MYI
          5          1 Version.MYD
      1,024            Version.MYI
```

This database has a total size of approximately 450 Megabytes.

Chapter 53

Bacula Security Issues

- Security means being able to restore your files, so read the Critical Items Chapter of this manual.
- The Clients (bacula-fd) must run as root to be able to access all the system files.
- It is not necessary to run the Director as root.
- It is not necessary to run the Storage daemon as root, but you must ensure that it can open the tape drives, which are often restricted to root access by default. In addition, if you do not run the Storage daemon as root, it will not be able to automatically set your tape drive parameters on most OSes since these functions, unfortunately require root access.
- You should restrict access to the Bacula configuration files, so that the passwords are not world-readable. The **Bacula** daemons are password protected using CRAM-MD5 (i.e. the password is not sent across the network). This will ensure that not everyone can access the daemons. It is a reasonably good protection, but can be cracked by experts.
- If you are using the recommended ports 9101, 9102, and 9103, you will probably want to protect these ports from external access using a firewall and/or using tcp wrappers (etc/hosts.allow).
- By default, all data that is sent across the network is unencrypted. However, Bacula does support TLS and can encrypt transmitted data. Please read the TLS (SSL) Communications Encryption section of this manual.
- You should ensure that the Bacula working directories are readable and writable only by the Bacula daemons.
- If you are using MySQL it is not necessary for it to run with **root** permission.
- The default Bacula grant-mysql-permissions script grants all permissions to use the MySQL database without a password. If you want security, please tighten this up!
- Don't forget that Bacula is a network program, so anyone anywhere on the network with the console program and the Director's password can access Bacula and the backed up data.
- You can restrict what IP addresses Bacula will bind to by using the appropriate **DirAddress**, **FDAddress**, or **SDAddress** records in the respective daemon configuration files.
- Be aware that if you are backing up your database using the default script, if you have a password on your database, it will be passed as a command line option to that script, and any user will be able to see this information. If you want it to be secure, you will need to pass it by an environment variable or a secure file.

  See also Backing Up Your Bacula Database — Security Considerations for more information.

53.1 Backward Compatibility

One of the major goals of Bacula is to ensure that you can restore tapes (I'll use the word tape to include disk Volumes) that you wrote years ago. This means that each new version of Bacula should be able to read old format tapes. The first problem you will have is to ensure that the hardware is still working some years down the road, and the second problem will be to ensure that the media will still be good, then your OS must be able to interface to the device, and finally Bacula must be able to recognize old formats. All the problems except the last are ones that we cannot solve, but by careful planning you can.

Since the very beginning of Bacula (January 2000) until today (December 2005), there have been two major Bacula tape formats. The second format was introduced in version 1.27 in November of 2002, and it has not changed since then. In principle, Bacula can still read the original format, but I haven't tried it lately so who knows ...

Though the tape format is fixed, the kinds of data that we can put on the tapes are extensible, and that is how we added new features such as ACLs, Win32 data, encrypted data, ... Obviously, an older version of Bacula would not know how to read these newer data streams, but each newer version of Bacula should know how to read all the older streams.

If you want to be 100% sure that you can read old tapes, you should:

1. Try reading old tapes from time to time – e.g. at least once a year.
2. Keep statically linked copies of every version of Bacula that you use in production then if for some reason, we botch up old tape compatibility, you can always pull out an old copy of Bacula ...

The second point is probably overkill but if you want to be sure, it may save you someday.

53.2 Configuring and Testing TCP Wrappers

TCP Wrappers are implemented if you turn them on when configuring (`./configure --with-tcp-wrappers`). With this code enabled, you may control who may access your daemons. This control is done by modifying the file: `/etc/hosts.allow`. The program name that **Bacula** uses when applying these access restrictions is the name you specify in the daemon configuration file (see below for examples). You must not use the **twist** option in your `/etc/hosts.allow` or it will terminate the Bacula daemon when a connection is refused.

The exact name of the package you need loaded to build with TCP wrappers depends on the system. For example, on SuSE, the TCP wrappers libraries needed to link Bacula are contained in the tcpd-devel package. On Red Hat, the package is named `tcp_wrappers`.

Dan Langille has provided the following information on configuring and testing TCP wrappers with Bacula.

If you read `hosts_options(5)`, you will see an option called twist. This option replaces the current process by an instance of the specified shell command. Typically, something like this is used:

```
ALL : ALL \
 : severity auth.info \
 : twist /bin/echo "You are not welcome to use %d from %h."
```

The libwrap code tries to avoid **twist** if it runs in a resident process, but that test will not protect the first `hosts_access()` call. This will result in the process (e.g. `bacula-fd`, `bacula-sd`,

bacula-dir) being terminated if the first connection to their port results in the twist option being invoked. The potential, and I stress potential, exists for an attacker to prevent the daemons from running. This situation is eliminated if your /etc/hosts.allow file contains an appropriate rule set. The following example is sufficient:

```
undef-fd : localhost : allow
undef-sd : localhost : allow
undef-dir : localhost : allow
undef-fd : ALL : deny
undef-sd : ALL : deny
undef-dir : ALL : deny
```

You must adjust the names to be the same as the Name directives found in each of the daemon configuration files. They are, in general, not the same as the binary daemon names. It is not possible to use the daemon names because multiple daemons may be running on the same machine but with different configurations.

In these examples, the Director is `undef-dir`, the Storage Daemon is `undef-sd`, and the File Daemon is `undef-fd`. Adjust to suit your situation. The above example rules assume that the SD, FD, and DIR all reside on the same box. If you have a remote FD client, then the following rule set on the remote client will suffice:

```
undef-fd : director.example.org : allow
undef-fd : ALL : deny
```

where director.example.org is the host which will be contacting the client (ie. the box on which the Bacula Director daemon runs). The use of "ALL : deny" ensures that the twist option (if present) is not invoked. To properly test your configuration, start the daemon(s), then attempt to connect from an IP address which should be able to connect. You should see something like this:

```
$ telnet undef 9103
Trying 192.168.0.56...
Connected to undef.example.org.
Escape character is '^]'.
Connection closed by foreign host.
$
```

This is the correct response. If you see this:

```
$ telnet undef 9103
Trying 192.168.0.56...
Connected to undef.example.org.
Escape character is '^]'.
You are not welcome to use undef-sd from xeon.example.org.
Connection closed by foreign host.
$
```

then twist has been invoked and your configuration is not correct and you need to add the deny statement. It is important to note that your testing must include restarting the daemons after each connection attempt. You can also use `btool`(8) and tcpdmatch(8) to validate your /etc/hosts.allow rules. Here is a simple test using tcpdmatch:

```
$ tcpdmatch undef-dir xeon.example.org
warning: undef-dir: no such process name in /etc/inetd.conf
client: hostname xeon.example.org
client: address 192.168.0.18
server: process undef-dir
matched: /etc/hosts.allow line 40
option: allow
access: granted
```

If you are running Bacula as a standalone daemon, the warning above can be safely ignored. Here is an example which indicates that your rules are missing a deny statement and the twist option has been invoked.

```
$ tcpdmatch undef-dir 10.0.0.1
warning: undef-dir: no such process name in /etc/inetd.conf
client: address 10.0.0.1
server: process undef-dir
matched: /etc/hosts.allow line 91
option: severity auth.info
option: twist /bin/echo "You are not welcome to use
  undef-dir from 10.0.0.1."
access: delegated
```

53.3 Running as non-root

Security advice from Dan Langille:

It is a good idea to run daemons with the lowest possible privileges. In other words, if you can, don't run applications as root which do not have to be root. The Storage Daemon and the Director Daemon do not need to be root. The File Daemon needs to be root in order to access all files on your system. In order to run as non-root, you need to create a user and a group. Choosing `bacula` as both the user name and the group name sounds like a good idea to me.

The FreeBSD port creates this user and group for you. Here is what those entries looked like on my FreeBSD laptop:

```
bacula:*:1002:1002::0:0:\bacula{} Daemon:/var/db/bacula:/sbin/nologin
```

I used `vipw` to create this entry. I selected a User ID and Group ID of 1002 as they were unused on my system.

I also created a group in `/etc/group`:

```
bacula:*:1002:
```

The `bacula` user (as opposed to the Bacula daemon) will have a home directory of `/var/db/bacula` which is the default location for the Bacula database.

Now that you have both a bacula user and a bacula group, you can secure the bacula home directory by issuing this command:

```
chown -R bacula:bacula /var/db/bacula/
```

This ensures that only the bacula user can access this directory. It also means that if we run the Director and the Storage daemon as bacula, those daemons also have restricted access. This would not be the case if they were running as root.

It is important to note that the storage daemon actually needs to be in the operator group for normal access to tape drives etc (at least on a FreeBSD system, that's how things are set up by default) Such devices are normally `chown root:operator`. It is easier and less error prone to make Bacula a member of that group than it is to play around with system permissions.

Starting the Bacula daemons

To start the Bacula daemons on a FreeBSD system, issue the following command:

```
/usr/local/etc/rc.d/bacula-dir start
/usr/local/etc/rc.d/bacula-sd  start
/usr/local/etc/rc.d/bacula-fd  start
```

To confirm they are all running:

```
$ ps auwx | grep bacula
root   63418 0.0 0.3 1856 1036 ?? Ss 4:09PM 0:00.00
    /usr/local/sbin/bacula-fd -v -c /usr/local/etc/bacula-fd.conf
bacula 63416 0.0 0.3 2040 1172 ?? Ss 4:09PM 0:00.01
    /usr/local/sbin/bacula-sd -v -c /usr/local/etc/bacula-sd.conf
bacula 63422 0.0 0.4 2360 1440 ?? Ss 4:09PM 0:00.00
    /usr/local/sbin/bacula-dir -v -c /usr/local/etc/bacula-dir.conf
```

Chapter 54

New Features in Older Bacula Enterprise Versions

This chapter presents some of the new features that have been added to the older Enterprise versions of Bacula. These features are available only with a Bacula Systems subscription.

54.1 Bacula Enterprise 8.8

Cloud Backup

A major problem of Cloud backup is that data transmission to and from the Cloud is very slow compared to traditional backup to disk or tape. The Bacula Cloud drivers provide a means to quickly finish the backups and then to transfer the data from the local cache to the Cloud in the background. This is done by first splitting the data Volumes into small parts that are cached locally, and then uploading those parts to the Cloud storage service in the background, either while the job continues to run or after the backup job has terminated. Once the parts are written to the Cloud, they may either be left in the local cache for quick restores or they can be removed (truncate cache).

Cloud Volume Architecture

The picture 54.1 on the next page shows two Volumes (Volume0001 and Volume0002) with their parts in the cache. Below the cache, one can see that Volume0002 has be uploaded or synchronized with the Cloud.

Note : Normal Bacula disk Volumes are implemented as standard files that reside in the user-defined **Archive Directory**. On the other hand, Bacula Cloud Volumes are directories that reside in the user-defined **Archive Directory**. The directory contains the cloud Volume parts, implemented as numbered files (`part.1`, `part.2`, ...).

Cloud Restore

During a restore, if the needed parts are available in the local cache, they will immediately be used. Otherwise, they will be downloaded from cloud storage as needed. The restore starts with parts already in the local cache but will wait as needed for any part that must be downloaded. The download proceeds while the restore is running.

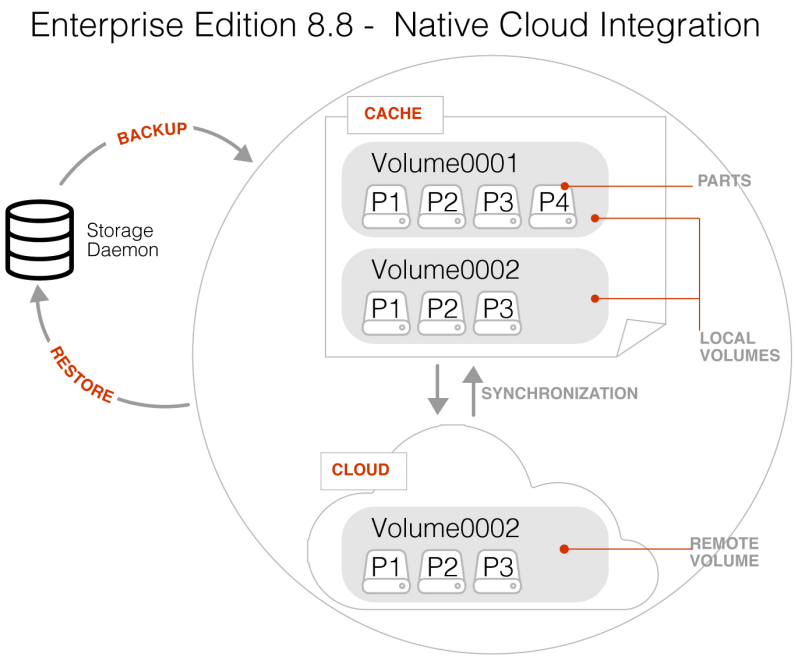


Figure 54.1: Bacula Cloud Architecture

With most cloud providers uploads are free of charge, but downloads of data from the cloud are billed. By using the local cache and multiple small parts, Bacula can be configured to substantially reduce download costs.

The **Maximum File Size** Device directive is valid within the Storage Daemon's cloud device configuration and defines the granularity of a restore chunk. In order to minimize the number of volume parts to download during a restore (in particular when restoring single files), it is useful to set the **Maximum File Size** to a value smaller than or equal to the configured **Maximum Part Size**.

Compatibility

Since a Cloud Volume contains the same data an ordinary Bacula Volume does, all existing types of Bacula data may be stored in the cloud – that is client data encryption, client-side compreion, plugin usage are all available. In fact, all existing Bacula functionality, with the exception of deduplication, is compatible with the Bacula Cloud drivers.

Deduplication and the Cloud

At the current time, Bacula Global Endpoint Backup does not support writing to the cloud because cloud storage would be too slow to support large hashed and indexed containers of deduplication data.

Virtual Autochangers and Disk Autochangers

Bacula Virtual Autochangers are compatible with the Bacula Cloud drivers. However, if you use a third party disk autochanger script such as **Vchanger**, unless or until it is modified to handle Volume directories, it may not be compatible with Bacula Cloud drivers.

Security

All data that is sent to and received from the cloud by default uses the HTTPS protocol, so data is encrypted while being transmitted and received. However, data that resides in the cloud is not encrypted by default. If that extra security of backed up data is required, Bacula's PKI data encryption feature should be used during the backup.

54.1.1 New Commands, Resource, and Directives for Cloud

To support Cloud storage devices, some new `bconsole` commands, new Storage Daemon directives, and a new Cloud resource that is referenced in the Storage Daemon's Device resource are available as of Bacula Enterprise 8.8

Cache and Pruning

The Cache is treated much like a normal Disk based backup, so that in configuring Cloud the administrator should take care to set **Archive Device** in the `Device` resource to a dirctory that would also be suitable for storing backup data. Obviously, unless the `truncate`/`prune` cache commands are used, the **Archive Device** will continue to fill.

The cache retention can be controlled per Volume with the **Cache Retention** attribute. The default value is 0, meaning that pruning of the cache is disabled.

The **Cache Retention** value for a volume can be modified with the `update` command, or configured via the `Pool` directive **Cache Retention** for newly created volumes.

New Cloud Bacula Console Commands

- `truncate cache`
- `upload`
- `cloud` The new cloud Bacula **Console** command allows inspecting and manipulating cloud volumes in different ways. The options are the following:
  - None. If you specify no arguments to the command, Bacula **Console** will prompt with:

    ```
    Cloud choice:
    1: List Cloud Volumes in the Cloud
    2: Upload a Volume to the Cloud
    3: Prune the Cloud Cache
    4: Truncate a Volume Cache
    5: Done
    Select action to perform on Cloud (1-5):
    ```

    The different choices should be rather obvious.
  - **truncate** This command will attempt to truncate the local cache for the specified Volume. Bacula will prompt you for the information needed to determine the Volume name or names. To avoid the prompts, the following additional command line options may be specified:
    - Storage=xxx
    - Volume=xxx
    - AllPools
    - AllFromPool
    - Pool=xxx
    - Storage=xxx

    - MediaType=xxx
    - Drive=xxx
    - Slots=nnn
  - `prune` This command will attempt to prune the local cache for the specified Volume. Bacula will respect the **Cache Retention** volume attribute to determine if the cache can be truncated or not. Only parts that are uploaded to the cloud will be deleted from the cache. Bacula will prompt you for the information needed to determine the Volume name or names. To avoid the prompts, the following additional command line options may be specified:
    - Storage=xxx
    - Volume=xxx
    - AllPools
    - AllFromPool
    - Pool=xxx
    - Storage=xxx
    - MediaType=xxx
    - Drive=xxx
    - Slots=nnn
  - `upload` This command will attempt to upload the specified Volumes. It will prompt for the information needed to determine the Volume name or names. To avoid the prompts, any of the following additional command line options can be specified:
    - Storage=xxx
    - Volume=xxx
    - AllPools
    - **AllFromPool**
    - Storage=xxx
    - Pool=xxx
    - MediaType=xxx
    - Drive=xxx
    - Slots=nnn
  - `list` This command will list volumes stored in the Cloud. If a volume name is specified, the command will list all parts for the given volume. To avoid the prompts, the operator may specify any of the following additional command line options:
    - Storage=xxx
    - Volume=xxx
    - Storage=xxx

Cloud Additions to the DIR `Pool` Resource

In `bacula-dir.conf` `Pool` resources, the directive Cache Retention can be specified. It is only effective for cloud storage backed volumes, and ignored when used with volumes stored on any different storage device.

Cloud Additions to the SD Device Resource

`Device` resource configured in the `bacula-sd.conf` file can use the Cloud keyword in the **Device Type** directive, and the two directives **Maximum Part Size** and **Cloud**.

New Cloud SD Device Directives

Device Type The Device Type has been extended to include the new keyword Cloud to specify that the device supports cloud Volumes. Example:

```
Device Type = Cloud
```

Cloud The new Cloud directive references a Cloud Resource. As with other Bacula resource references, the name of the Cloud resource is used as the value. Example:

```
Cloud = S3Cloud
```

Maximum Part Size This directive allows specification of the maximum size for each part of any volume written by the current device. Smaller part sizes will reduce restore costs, but will cause additional but small overhead to handle multiple parts. The maximum number of parts permitted per Cloud Volume is 524,288. The maximum size of any given part is approximately 17.5 TB.

Example Cloud Device Specification

An example of a Cloud Device Resource might be:

```
Device {
  Name = CloudStorage
  Device Type = Cloud
  Cloud = S3Cloud
  Archive Device = /opt/bacula/backups
  Maximum Part Size = 10000000
  Media Type = File
  LabelMedia = yes
  Random Access = Yes;
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
}
```

As can be seen above, the **Cloud** directive in the Device resource contains the name (**S3Cloud**), which references the Cloud resource that is shown below.

Note also that the **Archive Device** is specified in the same manner as used for a File device, i. e. by indicating a directory name. However, in place of containing regular files as Volumes, the archive device for the Cloud drivers will contain the local cache, which consists a directory per Volume, and these directories contain the parts associated with the particular Volume. So with the above Device resource, and the two cached Volumes shown in figure 54.1 on page 584 above, the following layout on disk would result:

```
/opt/bacula/backups
    /opt/bacula/backups/Volume0001
        /opt/bacula/backups/Volume0001/part.1
        /opt/bacula/backups/Volume0001/part.2
        /opt/bacula/backups/Volume0001/part.3
        /opt/bacula/backups/Volume0001/part.4
    /opt/bacula/backups/Volume0002
        /opt/bacula/backups/Volume0002/part.1
        /opt/bacula/backups/Volume0002/part.2
        /opt/bacula/backups/Volume0002/part.3
```

The Cloud Resource

The Cloud resource has a number of directives that may be specified as exemplified in the following example:

Default east USA location:

```
Cloud {
  Name = S3Cloud
  Driver = "S3"
  HostName = "s3.amazonaws.com"
  BucketName = "BaculaVolumes"
  AccessKey = "BZIXAIS39DP9YNER5DFZ"
  SecretKey = "beesheeg7iTe0Gaexee7aedie4aWohfuewohGaa0"
  Protocol = HTTPS
  URIStyle = VirtualHost
  Truncate Cache = No
  Upload = EachPart
  Region = ‘‘us-east-1"
  Maximum Upload Bandwidth = 5MB/s
}
```

For central europe location:

```
Cloud {
  Name = S3Cloud
  Driver = "S3"
  HostName = "s3-eu-central-1.amazonaws.com"
  BucketName = "BaculaVolumes"
  AccessKey = "BZIXAIS39DP9YNER5DFZ"
  SecretKey = "beesheeg7iTe0Gaexee7aedie4aWohfuewohGaa0"
  Protocol = HTTPS
  UriStyle = VirtualHost
  Truncate Cache = No
  Upload = EachPart
  Region = "eu-central-1"
  Maximum Upload Bandwidth = 4MB/s
}
```

For Amazon Cloud, refer to http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region to get a complete list of regions and corresponding endpoints and use them respectively as **Region** and **HostName** directives.

or in the following example for CEPH S3 interface:

```
  Cloud {
    Name = CEPH_S3
    Driver = "S3"
    HostName = ceph.mydomain.lan
    BucketName = "CEPHBucket"
    AccessKey = "xxxXXXxxxx"
    SecretKey = "xxheeg7iTe0Gaexee7aedie4aWohfuewohxx0"
    Protocol = HTTPS
    Upload = EachPart
    UriStyle = Path            # Must be set for CEPH
  }
```

For Azure:

```
Cloud {
  Name = MyCloud
  Driver = "Azure"
  HostName = "MyCloud" #not used but needs to be specified
  BucketName = "baculaAzureContainerName"
  AccessKey = "baculaaccess"
  SecretKey = "/Csw1SECRETUmZkfQ=="
  Protocol = HTTPS
  UriStyle = Path
}
```

The directives of the above `Cloud` resource for the S3 driver are defined as follows:

Name = <Device-Name> The name of the Cloud resource. This is the logical Cloud name, and may be any string up to 127 characters in length. Shown as **S3Cloud** above.

Description = <Text> The description is used for display purposes as is the case with all resources.

Driver = <Driver-Name> This defines which driver to use. At the moment, the only Cloud driver that is implemented is S3. There is also a File driver, which is used mostly for testing.

Host Name = <Name> This directive specifies the hostname to be used in the URL. Each Cloud service provider has a different and unique hostname. The maximum size is 255 characters and may contain a TCP port specification.

Bucket Name = <Name> This directive specifies the bucket name that you wish to use on the Cloud service. This name is normally a unique name that identifies where you want to place your Cloud Volume parts. With Amazon S3, the bucket must be created previously on the Cloud service. With Azure Storage, it is generaly refered as Container and it can be created automatically by Bacula when it does not exist. The maximum bucket name size is 255 characters.

Access Key = <String> The access key is your unique user identifier given to you by your cloud service provider.

Secret Key = <String> The secret key is the security key that was given to you by your cloud service provider. It is equivalent to a password.

Protocol = <HTTP | HTTPS> The protocol defines the communications protocol to use with the cloud service provider. The two protocols currently supported are: HTTPS and HTTP. The default is HTTPS.

Uri Style = <VirtualHost | Path> This directive specifies the URI style to use to communicate with the cloud service provider. The two **Uri Styles** currently supported are: VirtualHost and Path. The default is VirtualHost.

Truncate Cache = <truncate-kw> This directive specifies when Bacula should automatically remove (truncate) the local cache parts. Local cache parts can only be removed if they have been uploaded to the cloud. The currently implemented values are:

No Do not remove cache. With this option you must manually delete the cache parts with a `bconsole truncate cache`, or do so with an **Admin** Job that runs an `truncate cache` command. This is the default.

AfterUpload Each part will be removed just after it is uploaded. Note, if this option is specified, all restores will require a download from the Cloud[1].

AtEndOfJob With this option, at the end of the Job, every part that has been uploaded to the Cloud will be removed[2] (truncated).

Upload = <upload-kw> This directive specifies when local cache parts will be uploaded to the Cloud. The options are:

No Do not upload cache parts. With this option you must manually upload the cache parts with a Bacula **Console** `upload` command, or do so with an **Admin** Job that runs an `upload` command. This is the default.

EachPart With this option, each part will be uploaded when it is complete i.e. when the next part is created or at the end of the Job.

AtEndOfJob With this option all parts that have not been previously uploaded will be uploaded at the end of the Job.[3]

Maximum Concurrent Uploads = <number> The default is 3, but by using this directive, you may set it to any value you want.

[1] Not yet implemented
[2] Not yet implemented
[3] Not yet implemented

Maximum Concurrent Downloads = <number> The default is 3, but by using this directive, you may set it to any value you want.

Maximum Upload Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the upload bandwidth used globally by all devices referencing this Cloud resource.

Maximum Download Bandwidth = <speed> The default is unlimited, but by using this directive, you may limit the download bandwidth used globally by all devices referencing this Cloud resource.

Region = <String> The Cloud resource can be configured to use a specific endpoint within a region. This directive is required for AWS-V4 regions. ex: **Region**="eu-central-1"

BlobEndpoint = <String> This resource can be used to specify a custom URL for Azure Blob (see https://docs.microsoft.com/en-us/azure/storage/blobs/storage-custom-domain-name).

EndpointSuffix = <String> Use this resource to specify a custom URL postfix for Azure. ex: **EnbpointSuffix**="core.chinacloudapi.cn"

File Driver for the Cloud

As mentioned above, one may specify the keyword File on the **Driver** directive of the Cloud resource. Instead of writing to the Cloud, Bacula will instead create a Cloud Volume but write it to disk. The rest of this section applies to the Cloud resource directives when the File driver is specified.

The following Cloud directives are ignored: **Bucket Name**, **Access Key**, **Secret Key**, **Protocol**, **URI Style**. The directives **Truncate Cache** and **Upload** work on the local cache in the same manner as they do for the S3 driver.

The main difference to note is that the **Host Name**, specifies the destination directory for the Cloud Volume files, and this **Host Name** must be different from the **Archive Device** name, or there will be a conflict between the local cache (in the Archive Device directory) and the destination Cloud Volumes (in the Host Name directory).

As noted above, the File driver is mostly used for testing purposes, and we do not particularly recommend using it. However, if you have a particularly slow backup device you might want to stage your backup data into an SSD or disk using the local cache feature of the Cloud device, and have your Volumes transferred in the background to a slow File device.

54.1.2 Progressive Virtual Full

Instead of the implementation of Perpetual Virtual Full backups with a Perl script which needs to be run regularly, with Bacula Enterprise version 8.8.0, a new job directive named **Backups To Keep** has been added. This permits implementation of Progressive Virtual Fulls fully within Bacula itself.

To use the Progressive Virtual Full feature, the **Backups To Keep** directive is added to a Job resource. The value specified for the directive indicates the number of backup jobs that should not be merged into the Virtual Full. The default is zero and behaves the same way the prior script pvf worked.

54.1.3 Backups To Keep Directive

The new **BackupsToKeep** directive is specified in the Job Resource and has the form:

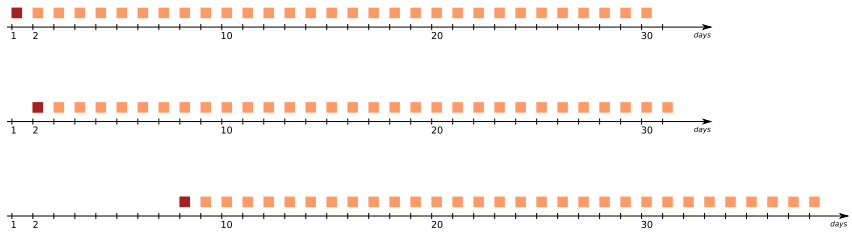


Figure 54.2: Backup Sequence Slides Forward One Day, Each Day

```
Backups To Keep = 30
```

where the value (30 in the figure 54.2) is the number of backups to retain. When this directive is present during a Virtual Full (it is ignored for any other Job types), Bacula will check if the latest Full backup has more subsequent backups than the value specified. In the above example, the Job would simply terminate unless there is a Full back followed by at least 31 backups of either Differential or Incremental level.

Assuming that the latest Full backup is followed by 32 Incremental backups, a Virtual Full will be run that consolidates the Full with the first two Incrementals that were run after the Full backup. The result is a Full backup followed by 30 Incremental ones. The Job Resource in `bacula-dir.conf` to accomplish this would be:

```
Job {
  Name = "VFull"
  Type = Backup
  Level = VirtualFull
  Client = "my-fd"
  File Set = "FullSet"
  Accurate = Yes
  Backups To Keep = 10
}
```

54.1.4 Delete Consolidated Jobs

The additional directive **Delete Consolidated Jobs** expects a <yes|no> value that, if set to yes, will cause any old Job that is consolidated during a Virtual Full to be deleted. In the example above we saw that a Full plus one other job (either an Incremental or Differential) were consolidated into a new Full backup. The original Full and the other Job consolidated would be deleted if this directive were set to yes. The default value is no.

54.1.5 Virtual Full Compatibility

Virtual Full as well as Progressive Virtual Full backups work with any standard backup Job including Jobs that use the Global Endpoint Deduplication.

However, it should be noted that Virtual Full jobs are **not compatible** with Windows backups using VSS writers (mostly plugins), nor are they compatible with a number of non-Windows Bacula Systems plugins. Please contact Bacula Systems Support team for more details Virtual Full compatibility.

54.1.6 TapeAlert Enhancements

There are some significant enhancements to the TapeAlert feature of Bacula. Several directives are used slightly differently, and there is a minor compatibility problem with the old TapeAlert implementation.

54.1.7 What is New

First, the **Alert Command** directive needs to be added in the Device resource that calls the new `tapealert` script that is installed in the scripts directory (normally: `/opt/bacula/scripts`):

```
Device {
  Name = ...
  Archive Device = /dev/nst0
  Alert Command = "/opt/bacula/scripts/tapealert %l"
  Control Device = /dev/sg1 # must be SCSI ctl for Archive Device
  ...
}
```

The **Control Device** directive in the Storage Daemon's configuration was previously used only for the SAN Shared Storage feature. With Bacula version 8.8, it is also used for the TapeAlert command to permit Bacula to detect tape alerts on a specific device (normally only tape devices).

Once the above mentioned two directives (**Alert Command** and **Control Device**) are in place in all `Device` resources, Bacula will check for tape alerts at two points:

- After the Drive is used and it becomes idle.
- After each read or write error on the drive.

At each of the above times, Bacula will call the new `tapealert` script, which uses the `tapeinfo` program. The `tapeinfo` utility is part of the `apt sg3-utils` and `rpm sg3\_utils` packages. Then for each tape alert that Bacula finds for that drive, it will emit a Job message that is either INFO, WARNING, or FATAL depending on the designation in the Tape Alert published by the https://www.t10.org[4]. For the specification, please see: `http://www.t10.org/ftp/t10/document.02/02-142r0.pdf`

As a somewhat extreme example, if tape alerts 3, 5, and 39 are set, you will get the following output in your backup job:

```
17-Nov 13:37 rufus-sd JobId 1: Error: block.c:287
Write error at 0:17 on device "tape"
(/home/kern/bacula/k/regress/working/ach/drive0)
Vol=TestVolume001.  ERR=Input/output error.

17-Nov 13:37 rufus-sd JobId 1: Fatal error: Alert:
Volume="TestVolume001" alert=3: ERR=The operation has stopped because
an error has occurred while reading or writing data which the drive
cannot correct.  The drive had a hard read or write error

17-Nov 13:37 rufus-sd JobId 1: Fatal error: Alert:
Volume="TestVolume001" alert=5: ERR=The tape is damaged or the drive
is faulty.  Call the tape drive supplier helpline.  The drive can no
longer read data from the tape

17-Nov 13:37 rufus-sd JobId 1: Warning: Disabled Device "tape"
(/home/kern/bacula/k/regress/working/ach/drive0) due to tape alert=39.

17-Nov 13:37 rufus-sd JobId 1: Warning: Alert: Volume="TestVolume001"
alert=39: ERR=The tape drive may have a fault.  Check for availability
```

[4] T10 Technical Committee on SCSI Storage Interfaces

```
of diagnostic information and run extended diagnostics if applicable.
The drive may have had a failure which may be identified by stored
diagnostic information or by running extended diagnostics (eg Send
Diagnostic).  Check the tape drive users manual for instructions on
running extended diagnostic tests and retrieving diagnostic data.
```

Without the tape alert feature enabled, you would only get the first error message above, which is the error Bacula received. Notice also, in this case the alert number 5 is a critical error, which causes two things to happen: First, the tape drive is disabled, and second, the Job is failed.

If you attempt to run another Job using the Device that has been disabled, you will get a message similar to the following:

```
17-Nov 15:08 rufus-sd JobId 2: Warning:
     Device "tape" requested by DIR is disabled.
```

and the Job may be failed if no other usable drive can be found.

Once the problem with the tape drive has been corrected, you can clear the tape alerts and re-enable the device with the Bacula Bacula **Console** command such as the following:

```
enable Storage=Tape
```

Note, when you enable the device, the list of prior tape alerts for that drive will be discarded.

Since is is possible to miss tape alerts, Bacula maintains a temporary list of the last 8 alerts, and each time Bacula calls the `tapealert` script, it will keep up to 10 alert status codes. Normally there will only be one or two alert errors for each call to the `tapealert` script.

Once a drive has one or more tape alerts, they can be inspected by using the Bacula **Console** status command as follows:

```
status storage=Tape
```

which produces the following output:

```
Device Vtape is "tape" (/home/kern/bacula/k/regress/working/ach/drive0)
mounted with:
    Volume:      TestVolume001
    Pool:        Default
    Media type:  tape
    Device is disabled. User command.
    Total Bytes Read=0 Blocks Read=1 Bytes/block=0
    Positioned at File=1 Block=0
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Hard Error
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Read Failure
    Warning Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
       alert=Diagnostics Required
```

If you want to see the long message associated with each of the alerts, simply set the debug level to 10 or more and re-issue the status command:

```
setdebug storage=Tape level=10
status storage=Tape
```

```
    ...
    Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
```

```
  flags=0x0 alert=The operation has stopped because an error has occurred
   while reading or writing data which the drive cannot correct. The drive had
   a hard read or write error
Critical Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001"
   flags=0x0 alert=The tape is damaged or the drive is faulty. Call the tape
   drive supplier helpline.  The drive can no longer read data from the tape
Warning Alert: at 17-Nov-2016 15:08:01 Volume="TestVolume001" flags=0x1
   alert=The tape drive may have a fault. Check for availability of diagnostic
   information and run extended diagnostics if applicable.   The drive may
   have had a failure which may be identified by stored diagnostic information
   or by running extended diagnostics (eg Send Diagnostic). Check the tape
   drive users manual for instructions on running extended diagnostic tests
   and retrieving diagnostic data.
...
```

The next time you **enable** the Device by either using Bacula **Console** or you restart the Storage Daemon, all the saved alert messages will be discarded.

54.1.8 Handling of Alerts

Tape Alerts numbered 7, 8, 13, 14, 20, 22, 52, 53, and 54 will cause Bacula to disable the current Volume.

Tape Alerts numbered 14, 20, 29, 30, 31, 38, and 39 will cause Bacula to disable the drive.

Please note certain tape alerts such as 14 have multiple effects (disable the Volume and disable the drive).

54.1.9 Multi-Tenancy Enhancements

54.1.10 New BWeb Management Suite Self User Restore

The BWeb Management Suite can be configured to allow authorized users to restore their own files on their own Unix or Linux system through BWeb. More information can be found in the BWeb Management Suite user's guide.

54.1.11 New Console ACL Directives

By default, if a **Console ACL** directive is not set, Bacula will assume that the ACL list is empty. If the current Bacula `Director` configuration uses restricted Consoles and allows restore jobs, it is mandatory to configure the new directives.

Directory ACL

This directive is used to specify a list of directories that can be accessed by a restore session. Without this directive, the console cannot restore any file. Multiple directories names may be specified by separating them with commas, and/or by specifying multiple **DirectoryACL** directives. For example, the directive may be specified as:

```
DirectoryACL = /home/bacula/, "/etc/", "/home/test/*"
```

With the above specification, the console can access the following files:

- `/etc/password`

- /etc/group
- /home/bacula/.bashrc
- /home/test/.ssh/config
- /home/test/Desktop/Images/something.png

But not the following files or directories:

- /etc/security/limits.conf
- /home/bacula/.ssh/id_dsa.pub
- /home/guest/something
- /usr/bin/make

If a directory starts with a Windows pattern (ex: c:/), Bacula will automatically ignore the case when checking directories.

UserId ACL

This directive is used to specify a list of UID/GID that can be accessed from a restore session. Without this directive, the console cannot restore any file. During the restore session, the Director will compute the restore list and will exclude files and directories that cannot be accessed. Bacula uses the LStat database field to retrieve st_mode, st_uid and st_gid information for each file and compare them with the **UserId ACL** elements. If a parent directory doesn't have a proper catalog entry, access to this directory will be automatically granted.

UID/GID names are resolved with getpwnam() function within the Director. The UID/GID mapping might be different from one system to an other.

Windows systems are **not compatible** with the **UserId ACL** feature. The use of **UserId ACL = *all*** is required to restore Windows systems from a restricted Console.

Multiple UID/GID names may be specified by separating them with commas, and/or by specifying multiple **UserId ACL** directives. For example, the directive may be specified as:

```
UserIdACL = "bacula", "100", "100:100", ":100", "bacula:bacula"
```

```
# ls -l /home
total 28
drwx------ 45 bacula bacula 12288 Oct 24 17:05 bacula
drwx------ 45 test   test   12288 Oct 24 17:05 test
drwx--x--x 45 test2  test2  12288 Oct 24 17:05 test2
drwx------  2 root   root   16384 Aug 30 14:57 backup
-rwxr--r--  1 root   root   1024  Aug 30 14:57 afile
```

In the example above, if the uid of the user test is 100, the following files will be accessible:

- bacula/*
- test/*
- test2/*

The directory backup will not be accessible.

54.1.12 Restore Job Security Enhancement

The Bacula **Console** `restore` command can now accept the new `jobuser=` and `jobgroup=` parameters to restrict the restore process to a given user account. Files and directories created during the restore session will be restricted.

```
* restore jobuser=joe jobgroup=users
```

The Restore Job restriction can be used on Linux and on FreeBSD. If the restore Client OS doesn't support the needed thread-level user impersonation, the restore job will be aborted.

54.1.13 New Bconsole "list" Command Behavior

The Bacula **Console** `list` commands can now be used safely from a restricted bconsole session. The information displayed will respect the ACL configured for the Console session. For example, if a Console has access to JobA, JobB and JobC, information about JobD will not appear in the `list jobs` command.

54.2 Bacula Enterprise 8.6.3

54.2.1 New Console ACL Directives

It is now possible to configure a restricted Console to distinguish Backup and Restore jobs permissions. The **Backup Client ACL** can restrict backup jobs on a specific set of clients, while the **Restore Client ACL** can restrict restore jobs.

```
# cat /opt/bacula/etc/bacula-dir.conf
...

Console {
 Name = fd-cons             # Name of the FD Console
 Password = yyy
...
 ClientACL = localhost-fd           # everything allowed
 RestoreClientACL = test-fd         # restore only
 BackupClientACL = production-fd    # backup only
}
```

The **Client ACL** directive takes precedence over the **Restore Client ACL** and the **Backup Client ACL** settings. In the `Console` resource above, this means that the bconsole linked to the `Console` named "fd-cons" will be able to run:

- backup and restore for "localhost-fd"
- backup for "production-fd"
- restore for "test-fd"

At restore time, jobs for client "localhost-fd", "test-fd" and "production-fd" will be available.

If *all* is set for **Client ACL**, backup and restore will be allowed for all clients, despite the use of **Restore Client ACL** or **Backup Client ACL**.

54.3 Bacula Enterprise 8.6.0

54.3.1 Client Initiated Backup

A console program such as the new `tray-monitor` or `bconsole` can now be configured to connect a `File Daemon`. There are many new features available (see the New Tray Monitor 54.3.3 on page 599), but probably the most important one is the ability for the user to initiate a backup of her own machine. The connection established by the FD to the `Director` for the backup can be used by the `Director` for the backup, thus not only can clients (users) initiate backups, but a `File Daemon` that is NATed (cannot be reached by the Director) can now be backed up without using advanced tunneling techniques.

The flow of information is shown in the picture 54.3

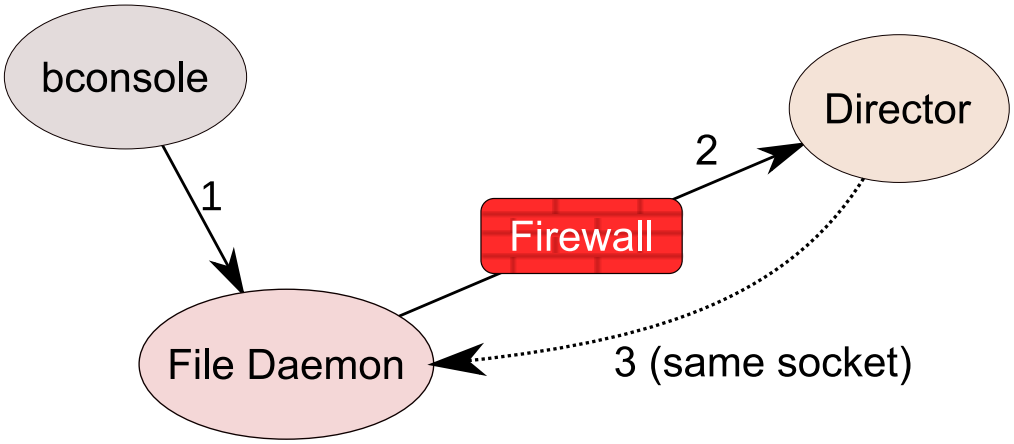


Figure 54.3: Client Initiated Backup Network Flow

54.3.2 Configuring Client Initiated Backup

In order to ensure security, there are a number of new directives that must be enabled in the new `tray-monitor`, the File Daemon and in the Director. A typical configuration might look like the following:

```
# cat /opt/bacula/etc/bacula-dir.conf
...

Console {
        Name = fd-cons             # Name of the FD Console
        Password = yyy

        # These commands are used by the tray-monitor, it is possible to restrict
        CommandACL = run, restore, wait, .status, .jobs, .clients
        CommandACL = .storages, .pools, .filesets, .defaults, .info

        # Adapt for your needs
        jobacl = *all*
        poolacl = *all*
        clientacl = *all*
        storageacl = *all*
        catalogacl = *all*
        filesetacl = *all*
}
```

```
# cat /opt/bacula/etc/bacula-fd.conf
...

Console {              # Console to connect the Director
        Name = fd-cons
        DIRPort = 9101
        address = localhost
        Password = "yyy"
        }

Director {
         Name = remote-cons   # Name of the tray monitor/bconsole
         Password = "xxx"     # Password of the tray monitor/bconsole
         Remote = yes         # Allow to use send commands to the Console defined
}
```

```
cat /opt/bacula/etc/bconsole-remote.conf
....

Director {
         Name = localhost-fd
         address = localhost        # Specify the FD address
         DIRport = 9102             # Specify the FD Port
         Password = "notused"
}

Console {
        Name = remote-cons         # Name used in the auth process
        Password = "xxx"
}
```

```
cat ~/.bacula-tray-monitor.conf

Monitor {
        Name = remote-cons
}

Client {
       Name = localhost-fd
       address = localhost     # Specify the FD address
```

```
        Port = 9102             # Specify the FD Port
        Password = "xxx"
        Remote = yes
}
```

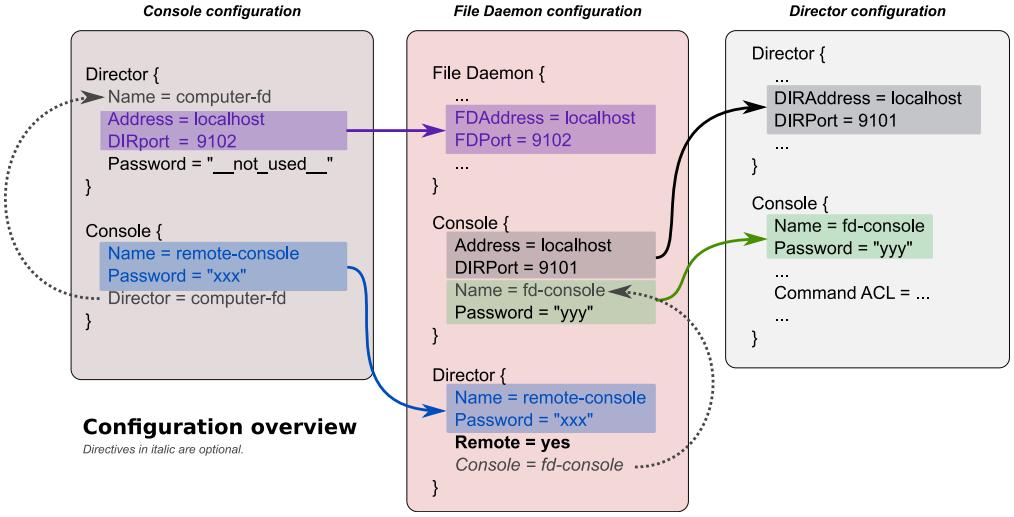


Figure 54.4: Relation Between Resources (bconsole)

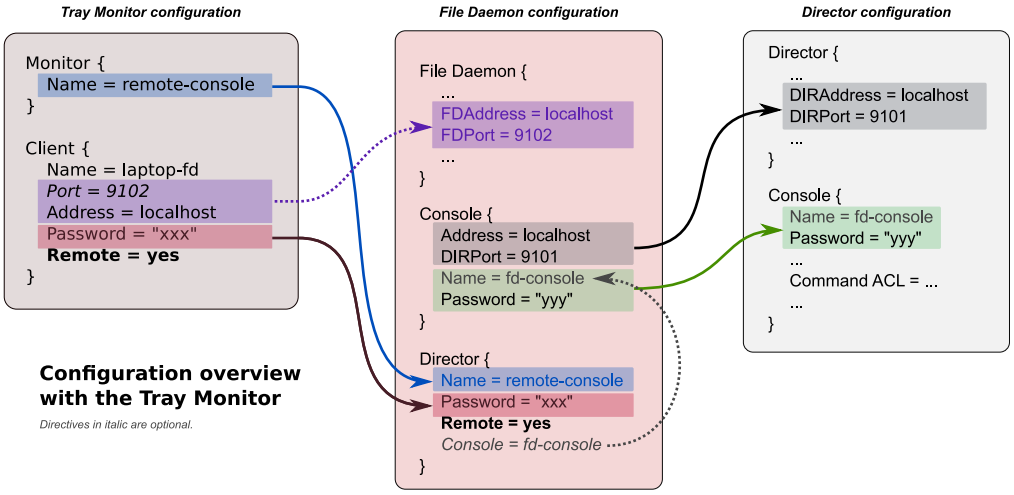


Figure 54.5: Relation Between Resources (tray-monitor)

A more detailed description with complete examples is available in the Tray monitor chapter of this manual.

54.3.3 New Tray Monitor

A new tray monitor has been added to the 8.6 release, which offers the following features:

- Director, File and Storage Daemon status page
- Support for the Client Initiated Backup protocol (See 54.3.1 on page 597). To use the Client Initiated Backup option from the tray monitor, the Client option "Remote" should be checked in the configuration (Fig. 54.7 on page 601).
- Wizard to run new job (Fig. 54.9 on page 602)

- Display an estimation of the number of files and the size of the next backup job (Fig. 54.9 on page 602)
- Ability to configure the tray monitor configuration file directly from the GUI (Fig. 54.7 on the facing page)
- Ability to monitor a component and adapt the tray monitor task bar icon if a jobs are running.
- TLS Support
- Better network connection handling
- Default configuration file is stored under `\$HOME/.bacula-tray-monitor.conf`
- Ability to "schedule" jobs
- Available for Linux and Windows platforms

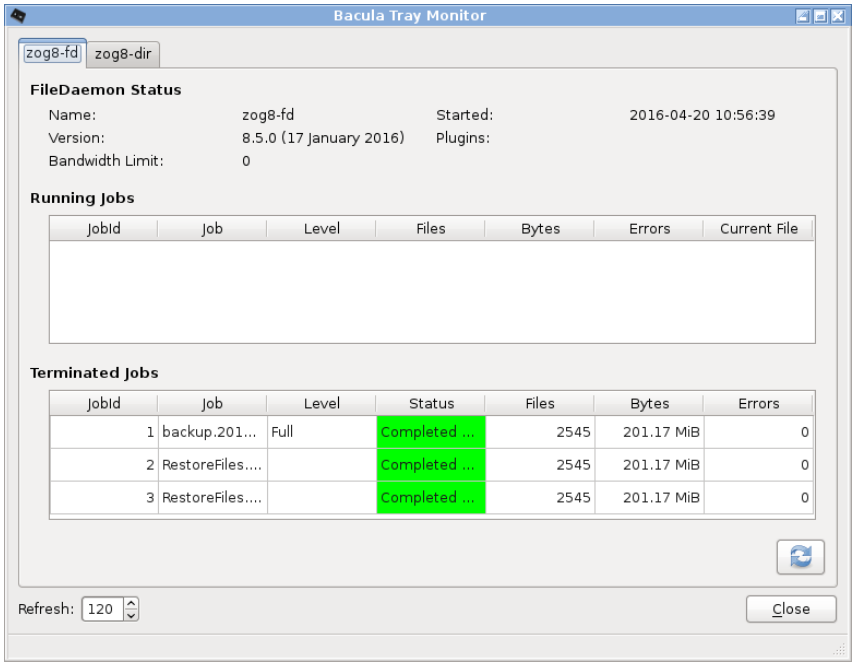


Figure 54.6: Tray Monitor Status

54.3.4 Scheduling Jobs via the Tray Monitor

The Tray Monitor can periodically scan a specific directory configured as **Command Directory** and process "`*.bcmd`" files to find jobs to run.

The format of the "`file.bcmd`" command file is the following:

```
<component name>:<run command>
<component name>:<run command>
...

<component name> = string
<run command>    = string (bconsole command line)
```

For example:

```
localhost-fd: run job=backup-localhost-fd level=full
localhost-dir: run job=BackupCatalog
```

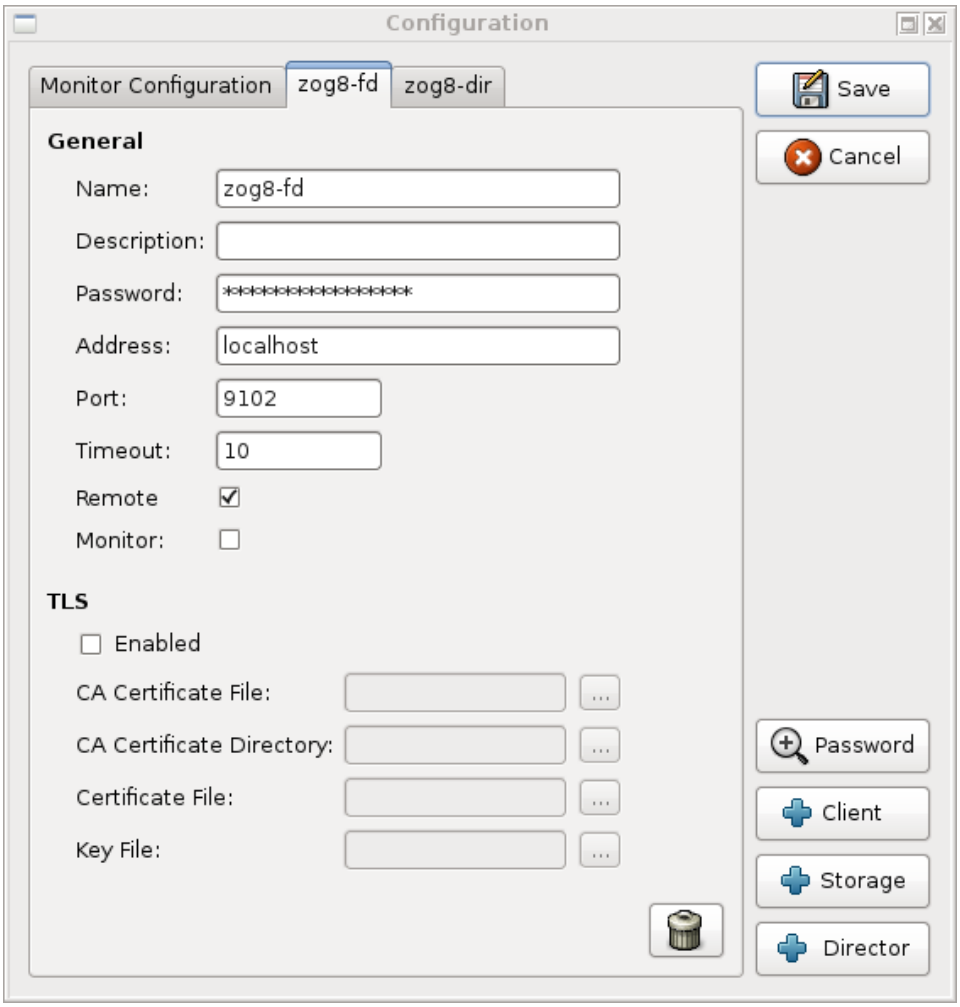


Figure 54.7: Tray Client Configuration

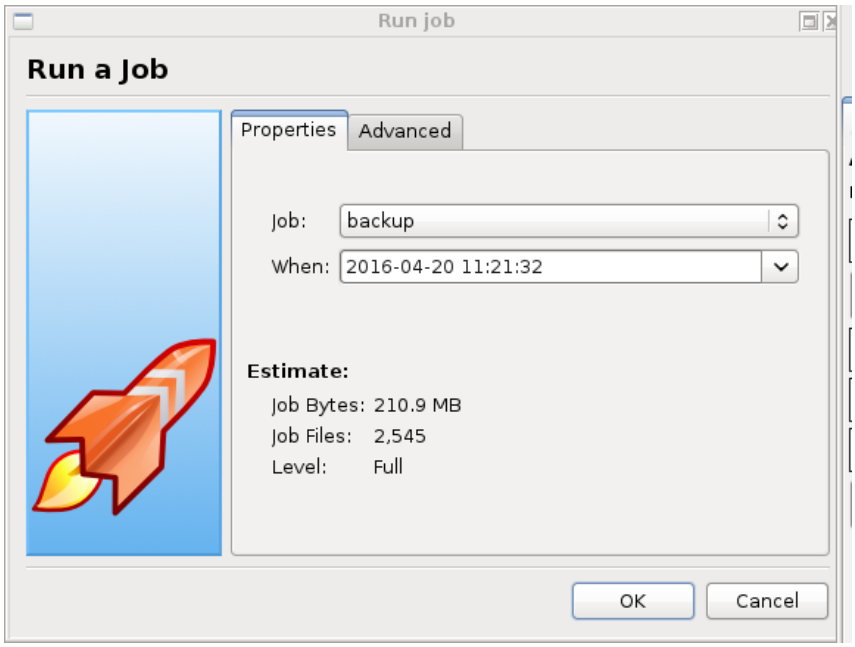


Figure 54.8: Tray Monitor Run a Job (1)

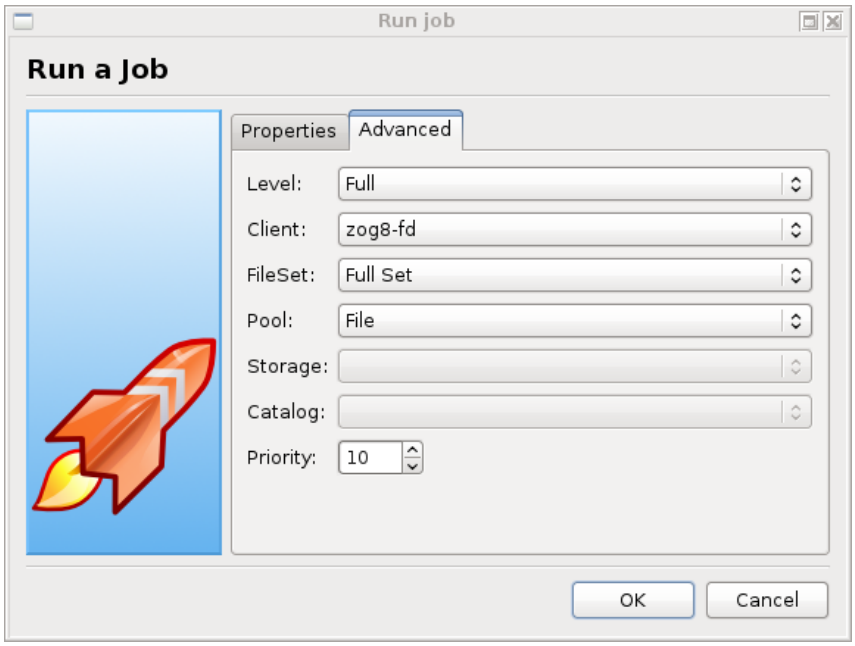


Figure 54.9: Tray Monitor Run a Job (2)

A command file should contain at least one command. The component specified in the first part of the command line should be defined in the tray monitor. Once the command file is detected by the tray monitor, a popup is displayed to the user and it is possible for the user to cancel the job.

Command files can be created with tools such as `cron` or the `task scheduler` on Windows. It is possible and recommended to verify network connectivity at that time to avoid network errors:

```
#!/bin/sh
if ping -c 1 director &> /dev/null
then
   echo "my-dir: run job=backup" > /path/to/commands/backup.bcmd
fi
```

54.3.5 Concurrent VSS Snapshot Support

It is now possible to run multiple concurrent jobs that use VSS snapshots on the File Daemon for Microsoft Windows.

54.3.6 Accurate Option for Verify "Volume Data" Job

As of Bacula version 8.4.1, it has been possible to have a Verify Job configured with **level**=Data that will reread all records from a job and optionally check size and checksum of all files.

Starting with 8.6, it is now possible to use the **accurate** option to check catalog records at the same time. Using a Verify job with **level**=Data and **accurate**=yes can replace the **level**=VolumeToCatalog option.

For more information on how to setup a Verify Data job, see 54.5.1 on page 607.

To run a Verify Job with the **accurate** option, it is possible to set the option in the Job definition or set use the **accurate**=yes on the command line.

```
* run job=VerifyData jobid=10 accurate=yes
```

54.3.7 Single Item Restore Optimisation

Bacula version 8.6.0 can generate indexes stored in the catalog to speed up file access during a Single Item Restore session for VMWare or for Exchange. The index can be displayed in bconsole with the `list filemedia` command.

```
* list filemedia jobid=1
```

54.3.8 FileDaemon Saved Messages Resource Destination

It is now possible to send the list of all saved files to a `Messages` resource with the saved message type. It is not recommended to send this flow of information to the `Director` and/or the `Catalog` when the client `FileSet` is large. To avoid side effects, the all keyword doesn't include the saved message type. The saved message type should be explicitly set.

```
# cat /opt/bacula/etc/bacula-fd.conf
...
Messages {
         Name = Standard
         director = mydirector-dir = all, !terminate, !restored, !saved
         append = /opt/bacula/working/bacula-fd.log = all, saved, restored
}
```

54.3.9 BWeb New Features

The 8.6 release adds some new BWeb features, such as:

- Two sets of wizards to help users to configure Copy/Migration jobs (Figures 54.11 on the following page and 54.12 on page 605)
- A wizard to run jobs (Fig. 54.13 on page 605)
- SSH integration in BWeb Security Center to restart components remotely (Fig. 54.14 on page 606)
- Global Endpoint Deduplication Overview screen (Fig. 54.10 on the following page)

54.3.10 Minor Enhancements

New Bconsole `.estimate` Command

The new `.estimate` command can be used to get statistics about a job to run. The command uses the database to estimate the size and the number of files of the next job. On a `PostgreSQL` database, the command uses regression slope to compute values. On `SQLite` or `MySQL`, where these statistical functions are not available, the command uses a simple "average" estimation. The correlation number is given for each value.

```
*.estimate job=backup
level=I
nbjob=0
```

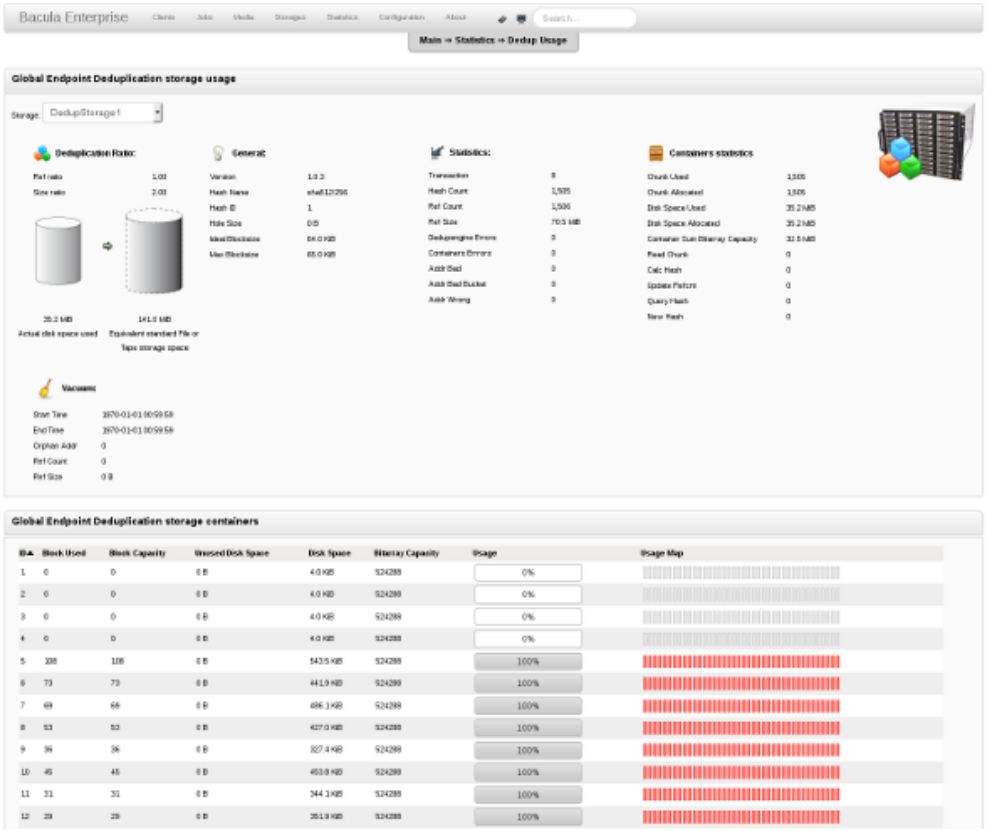


Figure 54.10: BWeb Global Endpoint Deduplication Overview

Figure 54.11: Copy Job Creation Wizard

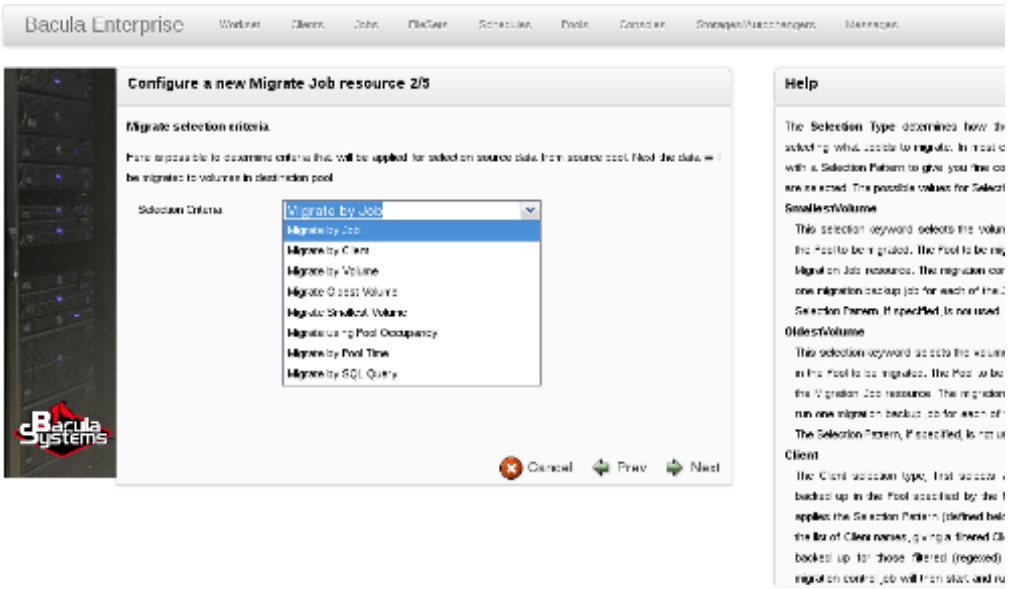


Figure 54.12: Migrate Job Creation Wizard

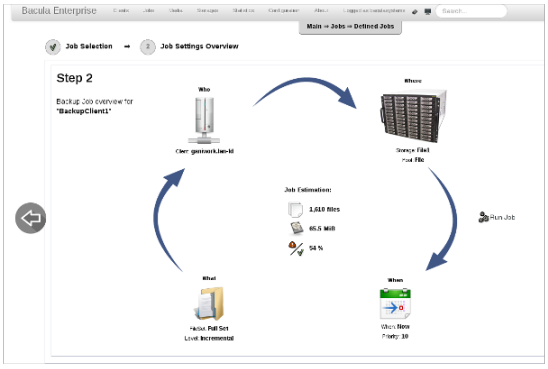


Figure 54.13: Run Job Wizard

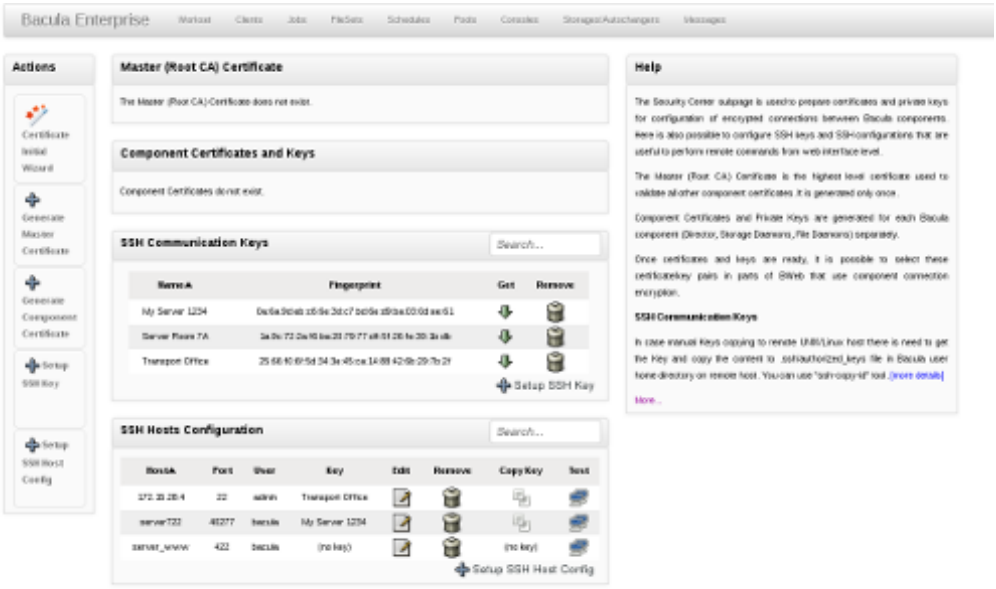


Figure 54.14: BWeb SSH Remote Control

```
corrbytes=0
jobbytes=0
corrfiles=0
jobfiles=0
duration=0
job=backup

*.estimate job=backup level=F
level=F
nbjob=1
corrbytes=0
jobbytes=210937774
corrfiles=0
jobfiles=2545
duration=0
job=backup
```

Traceback and Lockdump

After the reception of a signal by any of the Bacula daemon binaries, `traceback` and `lockdump` information are now stored in the same file.

54.4 Bacula Enterprise 8.4.10

54.4.1 Plugin for Microsoft SQL Server

A plugin for Microsoft SQL Server (`MSSQL`) is now available. The plugin uses `MSSQL` advanced backup and restore features (like Point In Time Recovery, Log backup, Differential backup, . . .).

```
Job {
    Name = MSSQLJob
    Type = Backup
    Client = windows1
```

```
    FileSet = MSSQL
    Pool = 1Month
    Storage = File
    Level = Incremental
}

FileSet {
        Name = MSSQL
        Enable VSS = no
        Include {
                Options {
                        Signature = MD5
                }
                Plugin = "mssql"
        }
}

FileSet {
        Name = MSSQL2
        Enable VSS = no
        Include {
                Options {
                        Signature = MD5
                }
                Plugin = "mssql: database=production"
        }
}
```

54.5 Bacula Enterprise 8.4.1

54.5.1 Verify Volume Data

It is now possible to have a Verify Job configured with `level=Data` to reread all records from a job and optionally check the size and the checksum of all files.

```
# Verify Job definition
Job {
    Name = VerifyData
    Type = Verify
    Level = Data
    Client = 127.0.0.1-fd     # Use local file daemon
    FileSet = Dummy           # Will be adapted during the job
    Storage = File            # Should be the right one
    Messages = Standard
    Pool = Default
}

# Backup Job definition
Job {
  Name = MyBackupJob
  Type = Backup
  Client = windows1
  FileSet = MyFileSet
  Pool = 1Month
  Storage = File
}

FileSet {
  Name = MyFileSet
  Include {
    Options {
      Verify = s5
      Signature = MD5
    }
  File = /
}
```

To run the Verify job, it is possible to use the "jobid" parameter of the run command.

```
*run job=VerifyData jobid=10
Run Verify Job
JobName:     VerifyData
Level:       Data
Client:      127.0.0.1-fd
FileSet:     Dummy
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  MyBackupJob.2015-11-11_09.41.55_03
Verify List: /opt/bacula/working/working/VerifyVol.bsr
When:        2015-11-11 09:47:38
Priority:    10
OK to run? (yes/mod/no): yes
Job queued. JobId=14

...

11-Nov 09:46 my-dir JobId 13: Bacula Enterprise 8.4.1 (13Nov15):
  Build OS:               x86_64-unknown-linux-gnu archlinux
  JobId:                  14
  Job:                    VerifyData.2015-11-11_09.46.29_03
  FileSet:                MyFileSet
  Verify Level:           Data
  Client:                 127.0.0.1-fd
  Verify JobId:           10
  Verify Job:q
  Start time:             11-Nov-2015 09:46:31
  End time:               11-Nov-2015 09:46:32
  Files Expected:         1,116
  Files Examined:         1,116
  Non-fatal FD errors:    0
  SD Errors:              0
  FD termination status:  Verify differences
  SD termination status:  OK
  Termination:            Verify Differences
```

The current Verify Data implementation requires specifying the correct Storage resource in the Verify job. The Storage resource can be changed with the bconsole command line and with the menu.

54.5.2 Bconsole list jobs Command Options

The list jobs bconsole command now accepts new command line options:

- **joberrors** Display jobs with JobErrors
- **jobstatus=T** Display jobs with the specified status code
- **client=client-name** Display jobs for a specified client
- **order=asc/desc** Change the output format of the job list. The jobs are sorted by start time and JobId, the sort can use ascending (asc) or descending (desc) order, the latter being the default.

54.5.3 Minor Enhancements

New Bconsole tee all Command

The "@tall" command allows logging all input and output of a console session.

```
*@tall /tmp/log
*st dir
```

```
...
@tall
```

MySQL Plugin Restore Options

It is now possible to specify the database name during a restore in the Plugin Option menu. It is still possible to use the "Where" parameter to specify the target database name.

PostgreSQL Plugin

We added a "timeout" option to the PostgreSQL plugin command line that is set to 60s by default. Users may want to change this value when the PostgreSQL cluster is slow to complete SQL queries used during the backup.

54.6 Bacula Enterprise 8.4

54.6.1 VMWare Single File Restore

It is now possible to explore VMWare virtual machines backup jobs (Full, Incremental and Differential) made with the Bacula Enterprise vSphere plugin to restore individual files and directories. The Single Item Restore feature comes with both a console interface and a BWeb Management Suite specific interface. See the VMWare Single File Restore whitepaper for more information.

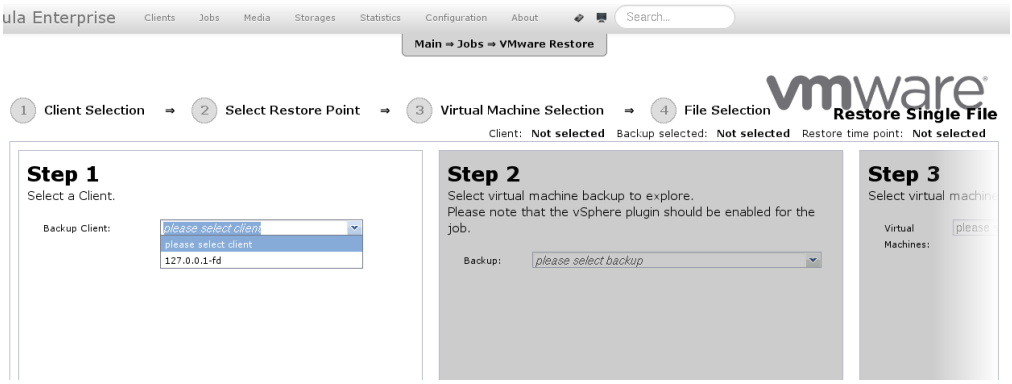


Figure 54.15: BWeb VMWare Single File Restore

54.6.2 Microsoft Exchange Single MailBox Restore

It is now possible to explore Microsoft Exchange databases backups made with the Bacula Enterprise VSS plugin to restore individual mailboxes. The Single Item Restore feature comes with both a console interface and a web interface. See the Exchange Single Mailbox Restore whitepaper for more information.

54.7 Bacula Enterprise 8.2.8

54.7.1 New Job Edit Codes %I

In various places such as RunScripts, you have now access to %I to get the JobId of the copy or migration job started by a migrate job.

```
Job {
  Name = Migrate-Job
  Type = Migrate
  ...
  RunAfter = "echo New JobId is %I"
}
```

54.8 Bacula Enterprise 8.2.2

54.8.1 New Job Edit Codes %E %R

In various places such as RunScripts, you have now access to %E to get the number of non-fatal errors for the current Job and %R to get the number of bytes read from disk or from the network during a job.

54.8.2 Enable/Disable commands

The bconsole enable and disable commands have been extended from enabling/disabling Jobs to include Clients, Schedule, and Storage devices. Examples:

```
disable Job=NightlyBackup Client=Windows-fd
```

will disable the Job named **NightlyBackup** as well as the client named **Windows-fd**.

```
disable Storage=LTO-changer Drive=1
```

will disable the first drive in the autochanger named **LTO-changer**.

Please note that doing a reload command will set any values changed by the enable/disable commands back to the values in the bacula-dir.conf file.

The Client and Schedule resources in the bacula-dir.conf file now permit the directive **Enabled** = yes or **Enabled** = no.

54.9 Bacula Enterprise 8.2

54.9.1 Snapshot Management

Bacula Enterprise 8.2 is now able to handle Snapshots on Linux/Unix systems. Snapshots can be automatically created and used to backup files. It is also possible to manage Snapshots from Bacula's bconsole tool through a unique interface.

Snapshot Backends

The following Snapshot backends are supported with Bacula Enterprise 8.2:

- BTRFS
- ZFS
- LVM[5]

By default, Snapshots are mounted (or directly available) under .snapshots directory on the root filesystem. (On ZFS, the default is .zfs/snapshots).

The Snapshot backend program is called bsnapshot and is available in the **bacula-enterprise-snapshot** package. In order to use the Snapshot Management feature, the package must be installed on the Client.

The bsnapshot program can be configured using /opt/bacula/etc/bsnapshot.conf file. The following parameters can be adjusted in the configuration file:

- trace=<file> Specify a trace file
- debug=<num> Specify a debug level
- sudo=<yes|no> Use sudo to run commands
- disabled=<yes|no> Disable snapshot support
- retry=<num> Configure the number of retries for some operations
- snapshot_dir=<dirname> Use a custom name for the Snapshot directory. (.SNAPSHOT, .snapdir, etc.)
- lvm_snapshot_size=<lvpath:size> Specify a custom snapshot size for a given LVM volume
- mountopts=<devpath:options> Specify a custom mount option for a give device (avaiable in 10.0.4)

```
# cat /opt/bacula/etc/bsnapshot.conf
trace=/tmp/snap.log
debug=10
lvm_snapshot_size=/dev/ubuntu-vg/root:5%
mountopts=nouuid
mountopts=/dev/ubuntu-vg/root:nouuid,nosuid
```

Application Quiescing

When using Snapshots, it is very important to quiesce applications that are running on the system. The simplest way to quiesce an application is to stop it. Usually, taking the Snapshot is very fast, and the downtime is only about a couple of seconds. If downtime is not possible and/or the application provides a way to quiesce, a more advanced script can be used. An example is described on 54.9.1 on the following page.

New Director Directives

The use of the Snapshot Engine on the FileDaemon is determined by the new **Enable Snapshot** FileSet directive. The default is no.

[5]Some restrictions described in 54.9.1 on page 614 applies to the LVM backend

```
FileSet {
  Name = LinuxHome

  Enable Snapshot = yes

  Include {
    Options = { Compression = LZO }
    File = /home
  }
}
```

By default, Snapshots are deleted from the Client at the end of the backup. To keep Snapshots on the Client and record them in the Catalog for a determined period, it is possible to use the **Snapshot Retention** directive in the Client or in the Job resource. The default value is 0 secconds. If, for a given Job, both Client and Job **Snapshot Retention** directives are set, the Job directive will be used.

```
Client {
   Name = linux1
   ...

   Snapshot Retention = 5 days
}
```

To automatically prune Snapshots, it is possible to use the following RunScript command:

```
Job {
   ...
   Client = linux1
   ...
   RunScript {
      RunsOnClient = no
      Console = "prune snapshot client=%c yes"
      RunsAfter = yes
   }
}
```

In RunScripts, the `AfterSnapshot` keyword for the `RunsWhen` directive will allow a command to be run just after the Snapshot creation.

`AfterSnapshot` is a synonym for the `AfterVSS` keyword.

```
Job {
 ...
  RunScript {
    Command = "/etc/init.d/mysql start"
    RunsWhen = AfterSnapshot
    RunsOnClient = yes
  }
  RunScript {
    Command = "/etc/init.d/mysql stop"
    RunsWhen = Before
    RunsOnClient = yes
  }
}
```

Job Output Information

Information about Snapshots are displayed in the Job output. The list of all devices used by the Snapshot Engine is displayed, and the Job summary indicates if Snapshots were available.

```
JobId 3:    Create Snapshot of /home/build
JobId 3:    Create Snapshot of /home/build/subvol
JobId 3:    Delete snapshot of /home/build
JobId 3:    Delete snapshot of /home/build/subvol
...
JobId 3: Bacula 127.0.0.1-dir 8.2.0 (23Feb15):
  Build OS:               x86_64-unknown-linux-gnu archlinux
  JobId:                  3
  Job:                    Incremental.2015-02-24_11.20.27_08
  Backup Level:           Full
...
  Snapshot/VSS:           yes
...
  Termination:            Backup OK
```

New `snapshot` Bconsole Commands

The new `snapshot` command will display by default the following menu:

```
*snapshot
Snapshot choice:
     1: List snapshots in Catalog
     2: List snapshots on Client
     3: Prune snapshots
     4: Delete snapshot
     5: Update snapshot parameters
     6: Update catalog with Client snapshots
     7: Done
Select action to perform on Snapshot Engine (1-7):
```

The `snapshot` command can also have the following parameters:

```
[client=<client-name> | job=<job-name> | jobid=<jobid>]
 [delete | list | listclient | prune | sync | update]
```

It is also possible to use traditional `list`, `llist`, `update`, `prune` or `delete` commands on Snapshots.

```
*llist snapshot jobid=5
 snapshotid: 1
       name: NightlySave.2015-02-24_12.01.00_04
 createdate: 2015-02-24 12:01:03
     client: 127.0.0.1-fd
    fileset: Full Set
      jobid: 5
     volume: /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
     device: /home/btrfs
       type: btrfs
  retention: 30
    comment:
```

```
* snapshot listclient
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:9102
Snapshot      NightlySave.2015-02-24_12.01.00_04:
  Volume:     /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
  Device:     /home
  CreateDate: 2015-02-24 12:01:03
  Type:       btrfs
  Status:     OK
  Error:
```

With the *Update catalog with Client snapshots* option (or `snapshot sync`), the Director contacts the FileDaemon, lists snapshots of the system and creates catalog records of the Snapshots.

```
*snapshot sync
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:9102
Snapshot      NightlySave.2015-02-24_12.35.47_06:
  Volume:     /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
  Device:     /home
  CreateDate: 2015-02-24 12:35:47
  Type:       btrfs
  Status:     OK
  Error:
Snapshot added in Catalog

*llist snapshot
 snapshotid: 13
       name: NightlySave.2015-02-24_12.35.47_06
 createdate: 2015-02-24 12:35:47
     client: 127.0.0.1-fd
    fileset:
      jobid: 0
     volume: /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
     device: /home
       type: btrfs
  retention: 0
    comment:
```

LVM Backend Restrictions

LVM Snapshots are quite primitive compared to ZFS, BTRFS, NetApp and other systems. For example, it is not possible to use Snapshots if the Volume Group (VG) is full. The administrator must keep some free space in the VG to create Snapshots. The amount of free space required depends on the activity of the Logical Volume (LV). bsnapshot uses 10% of the LV by default. This number can be configured per LV in the bsnapshot.conf file (See 54.9.1 on page 611).

```
[root@system1]# vgdisplay
  --- Volume group ---
  VG Name               vg_ssd
...
  VG Size               29,81 GiB
...
  Alloc PE / Size       125 / 500,00 MiB
  Free  PE / Size       7507 / 29,32 GiB   <---- Free Space
...
```

It is also not advisable to leave snapshots on the LVM backend. Having multiple snapshots of the same LV on LVM will slow down the system.

Only Ext4, XFS and EXT3 [6] filesystems are supported with the Snapshot LVM backend.

Debug Options

To get low level information about the Snapshot Engine, the debug tag "snapshot" should be used in the setdebug command.

```
* setdebug level=10 tags=snapshot client
* setdebug level=10 tags=snapshot dir
```

[6] XFS and EXT3 are available in 8.2.7 and later

54.9.2 Global Endpoint Deduplication(TM)

Storage to Storage Copy/Migration

Copy and Migration Jobs now use the Global Endpoint Deduplication protocol if the destination Device Type is `dedup`.

Performance Enhancements

A new automatic Deduplication index optimization has been added to the Vacuum procedure.

Part of the Deduplication index can be locked into memory to improve performance.

Users can now configure parameters related to the size of the Deduplication index and the amount of memory that can be used to cache the index.

54.9.3 Hypervisor Plugins

Hyper-V VSS Plugin

Backing up and restoring Hyper-V virtual machines is supported with Full level backups using the VSS API. Use of the Global Endpoint Deduplication plugin and the `bothsides` FileSet option minimizes the amount of data transfered and the amount of storage used.

KVM Plugin

The KVM plugin provides the following main features:

- File level backup
- Automatic virtual machine discovery
- Full, Differential, Incremental backup level support
- The ability to handle inclusion/exclusion of files

The KVM plugin is designed to be used when the hypervisor uses local storage for virtual machine disks and `libvirtd` for virtual machine management.

54.9.4 Windows Encrypted File System (EFS) Support

The Bacula Enterprise Windows File Daemon now automatically supports files and directories that are encrypted on Windows filesystem.

54.9.5 BWeb Management Suite

54.9.6 Minor Enhancements

Copy/Migration/VirtualFull Performance Enhancements

The Copy, Migration and VirtualFull performance on large jobs with millions of files has been greatly enhanced.

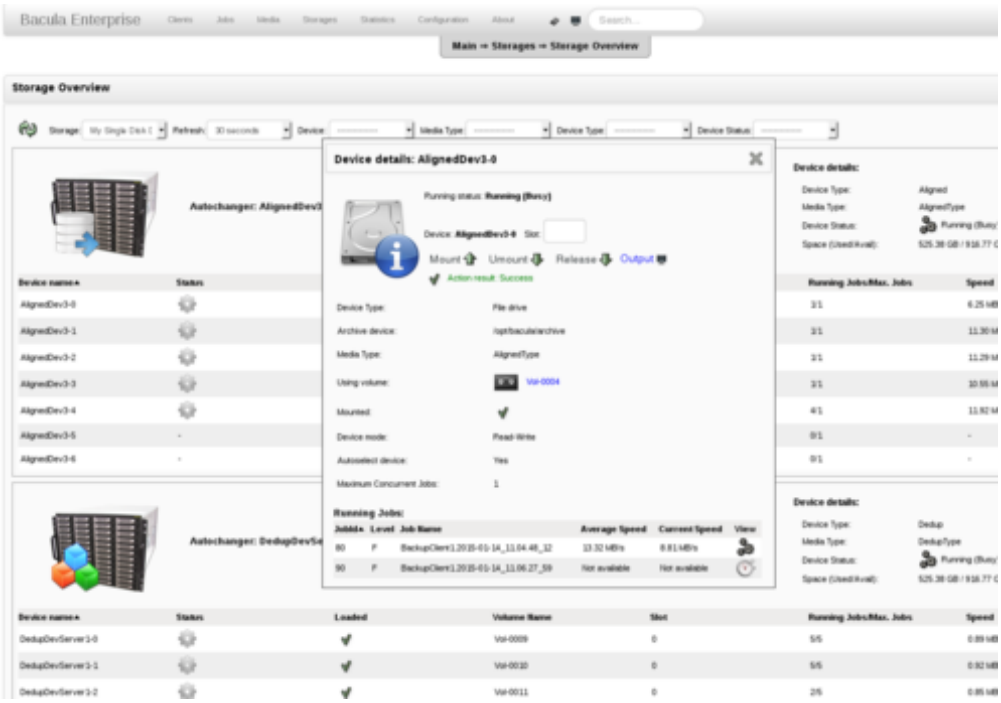


Figure 54.16: BWeb Storage Daemon Status/Overview

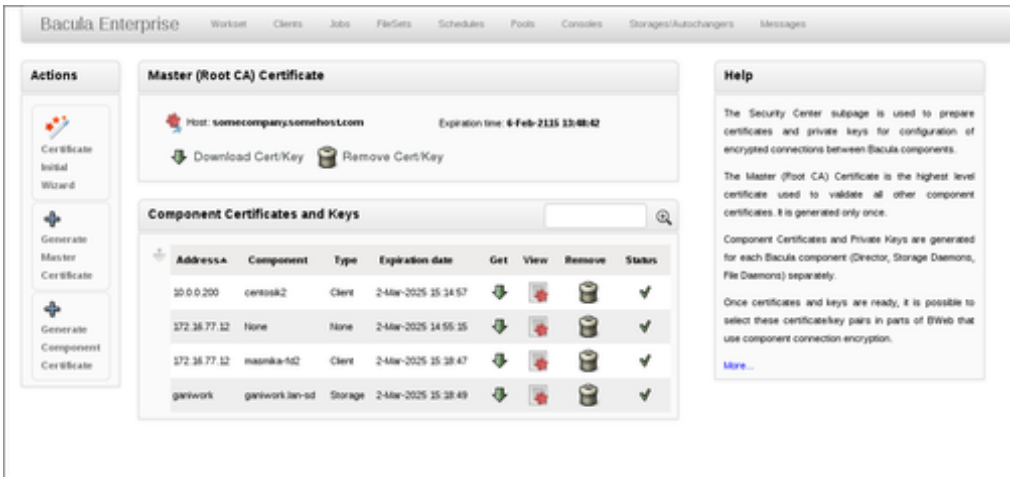


Figure 54.17: BWeb TLS Security Center

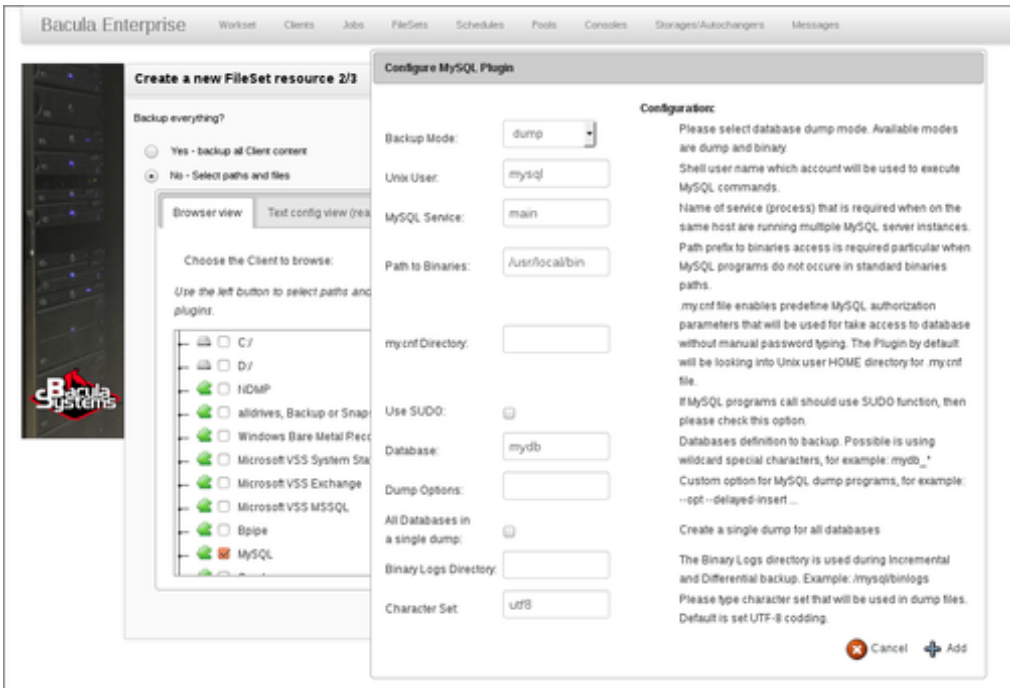


Figure 54.18: BWeb FileSet Wizard

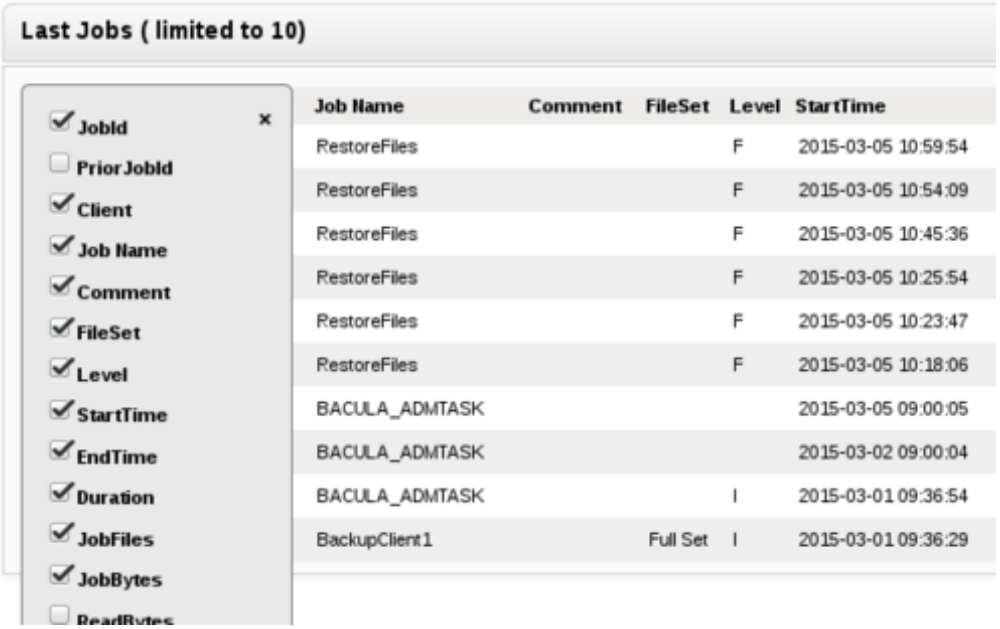


Figure 54.19: BWeb Job/Media Table Configuration

Storage Daemon Reports Disk Usage

The `status storage` command now reports the space available on disk devices:

```
...
Device status:

Device file: "FileStorage" (/bacula/arch1) is not open.
    Available Space=5.762 GB
==

Device file: "FileStorage1" (/bacula/arch2) is not open.
    Available Space=5.862 GB
```

54.10 Bacula Enterprise 8.0

54.10.1 Global Endpoint Deduplication™

The *Global Endpoint Deduplication* solution minimizes network transfers and Bacula Volume size using deduplication technology.

The new Global Endpoint Deduplication Storage daemon directives are:

Device Type = Dedup sets the Storage device for deduplication. Deduplication is performed only on disk volumes.

Dedup Directory = this directive specifies where the deduplicated blocks will be stored. Blocks that are deduplicated will be placed in this directory rather than in the Bacula Volume, which will only contain a reference pointer to the deduplicated blocks.

Dedup Index Directory in addition to the deduplicated blocks, when deduplication is enabled, the Storage daemon keeps an index of the deduplicated block locations. This index will be frequently consulted during the deduplication backup process, so it should be placed on the fastest device possible (e.g. an SSD).

See below for a FileSet example using the new dedup directive.

Configuration Example

In the Storage Daemon configuration file, you must define a Device with the **DeviceType = Dedup**. It is also possible to configure where the Storage Daemon will store blocks and indexes. Blocks will be stored in the **Dedup Directory**, the directory is common for all **Dedup** devices and should have a large amount of free space. Indexes will be stored in the **Dedup Index Directory**, indexes will have a lot of random update access, and can benefit from SSD drives.

```
# from bacula-sd.conf
 Storage {
   Name = my-sd
   Working Directory = /opt/bacula/working
   Pid Directory = /opt/bacula/working

   Plugin Directory = /opt/bacula/plugins
   Dedup Directory = /opt/bacula/dedup
   Dedup Index Directory = /opt/bacula/ssd  # default for Dedup Directory
 }

 Device {
```

```
    Name = DedupDisk
    Archive Device = /opt/bacula/storage
    Media Type = DedupVolume
    Label Media = yes
    Random Access = yes
    Automatic Mount = yes
    Removable Media = no
    Always Open = no

    Device Type = Dedup    # Required
  }
```

The Global Endpoint Deduplication Client cache system can speed up restore jobs by getting blocks from the local client disk instead of requesting them over the network. Note that if blocks are not available locally, the FileDaemon will get blocks from the Storage Daemon. This feature can be enabled with the **Dedup Index Directory** directive in the FileDaemon resource. When using this option, the File Daemon will have to maintain the cache during Backup jobs.

```
# from bacula-fd.conf
 FileDaemon {
   Name = my-fd
   Working Directory = /opt/bacula/working
   Pid Directory = /opt/bacula/working

   # Optional, Keep indexes on the client for faster restores
   Dedup Index Directory = /opt/bacula/dedupindex
 }
```

It is possible to configure the Global Endpoint Deduplication system in the Director with a FileSet directive called **Dedup**. Each FileSet Include section can specify a different deduplication behavior depending on your needs.

```
 FileSet {
   Name = FS_BASE

   # Send everything to the Storage Daemon as usual
   # and let the Storage Daemon do the deduplication
   Include {
     Options {
       Dedup = storage
     }
     File = /opt/bacula/etc
   }

   # Send only references and new blocks to the Storage Daemon
   Include {
     Options {
       Dedup = bothsides
     }
     File = /VirtualBox
   }

   # Do not try to dedup my encrypted directory
   Include {
     Options {
       Dedup = none
     }
     File = /encrypted
   }
 }
```

The FileSet **Dedup** directive accepts the following values:

* **storage** All the deduplication work is done on the SD side if the device type is dedup (default value). This option is useful if you want to avoid the extra client-side disk space overhead that will occur with the **bothsides** option.

- **none** Force FD and SD to not use deduplication
- **bothsides** The deduplication work is done on both the FD and the SD. Only references and new blocks will be transfered over the network.

54.10.2 Storage Daemon to Storage Daemon

Bacula Enterprise version 8.0 now permits SD to SD transfer of Copy and Migration Jobs. This permits what is commonly referred to as replication or off-site transfer of Bacula backups. It occurs automatically if the source SD and destination SD of a Copy or Migration job are different. That is, the SD to SD transfers need no additional configuration directives. The following picture shows how this works.

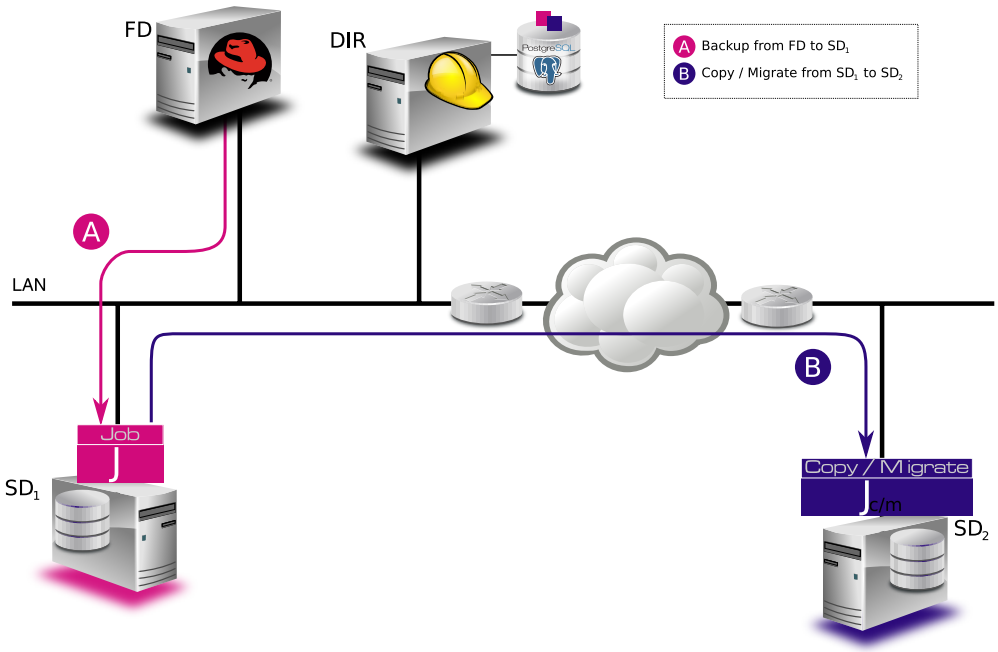


Figure 54.20: SD to SD

54.10.3 Windows Mountpoint Support

Bacula Enterprise version 8.0 is now able to detect Windows mountpoints and include volumes automatically in the VSS snapshot set. To backup all local disks on a Windows server, the following FileSet is now accepted. It depreciates the alldrives plugin.

```
FileSet {
 Name = "All Drives"
 Include {
   Options {
     Signature = MD5
   }

   File = /
 }
}
```

If you have mountpoints, the `onefs=no` option should be used as it is with Unix systems.

```
FileSet {
 Name = "All Drives with mountpoints"
 Include {
   Options {
     Signature = MD5
     OneFS = no
   }
   File = C:/                # will include mountpoint C:/mounted/...
 }
}
```

To exclude a mountpoint from a backup when `OneFS = no`, use the Exclude block as usual:

```
FileSet {
 Name = "All Drives with mountpoints"
 Include {
   Options {
     Signature = MD5
     OneFS = no
   }
   File = C:/           # will include all mounted mountpoints under C:/
                        # including C:/mounted  (see Exclude below)
 }

 Exclude {
   File = C:/mounted    # will not include C:/mounted
 }
}
```

54.10.4 SD Calls Client

If the **SD Calls Client** directive is set to true in a `Client` resource any Backup, Restore, Verify Job where the client is involved, the client will wait for the Storage daemon to contact it. By default this directive is set to false, and the Client will call the Storage daemon as it always has. This directive can be useful if your Storage daemon is behind a firewall that permits outgoing connections but not incoming connections. The picture 54.21 shows the communications connection paths in both cases.

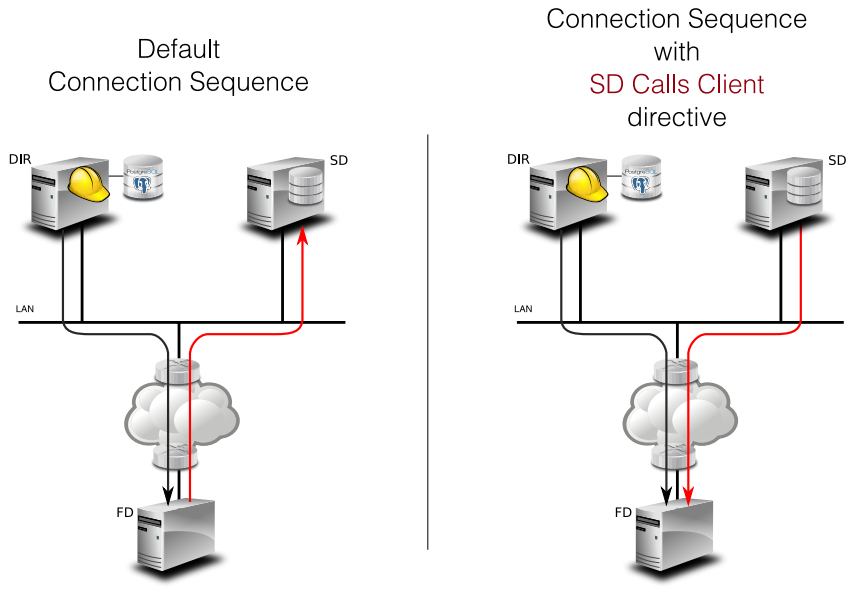


Figure 54.21: SD Calls Client

54.10.5 Data Encryption Cipher Configuration

Bacula Enterprise version 8.0 and later now allows configuration of the data encryption cipher and the digest algorithm. Previously, the cipher was forced to AES 128, but it is now possible to choose between the following ciphers:

- AES128 (default)
- AES192
- AES256
- blowfish

The digest algorithm was set to SHA1 or SHA256 depending on the local OpenSSL options. We advise you to not modify the PkiDigest default setting. Please, refer to the OpenSSL documentation to understand the pros and cons regarding these options.

```
FileDaemon {
    ...
    PkiCipher = AES256
}
```

54.10.6 Minor Enhancements

New Option Letter "M" for Accurate Directive in FileSet

Added in version 8.0.5, the new "M" option letter for the Accurate directive in the FileSet Options block, which allows comparing the modification time and/or creation time against the last backup timestamp. This is in contrast to the existing options letters "m" and/or "c", mtime and ctime, which are checked against the stored catalog values, which can vary accross different machines when using the BaseJob feature.

The advantage of the new "M" option letter for Jobs that refer to BaseJobs is that it will instruct Bacula to backup files based on the last backup time, which is more useful because the mtime/ctime timestamps may differ on various Clients, causing files to be needlessly backed up.

```
  Job {
    Name = USR
    Level = Base
    FileSet = BaseFS
...
  }

  Job {
    Name = Full
    FileSet = FullFS
    Base = USR
...
  }

  FileSet {
    Name = BaseFS
    Include {
      Options {
        Signature = MD5
      }
      File = /usr
    }
  }
```

```
FileSet {
  Name = FullFS
  Include {
    Options {
      Accurate = Ms      # check for mtime/ctime of last backup timestamp and Size
      Signature = MD5
    }
    File = /home
    File = /usr
  }
}
```

.api version 2

In Bacula Enterprise version 8.0 and later, we introduced a new .api version to help external tools to parse various Bacula bconsole output.

The `api_opts` option can use the following arguments:

C Clear current options

tn Use a specific time format (1 ISO format, 2 Unix Timestamp, 3 Default Bacula time format)

sn Use a specific separator between items (new line by default).

Sn Use a specific separator between objects (new line by default).

o Convert all keywords to lowercase and convert all non *isalpha* characters to _

```
  .api 2 api_opts=t1s43S35
  .status dir running
==================================
jobid=10
job=AJob
...
```

New Debug Options

In Bacula Enterprise version 8.0 and later, we introduced a new `options` parameter for the `setdebug` bconsole command.

The following arguments to the new `option` parameter are available to control debug functions.

0 Clear debug flags

i Turn off, ignore bwrite() errors on restore on File Daemon

d Turn off decomp of BackupRead() streams on File Daemon

t Turn on timestamps in traces

T Turn off timestamps in traces

c Truncate trace file if trace file is activated

l Turn on recoding events on P() and V()

p Turn on the display of the event ring when doing a backtrace

The following command will enable debugging for the File Daemon, truncate an existing trace file, and turn on timestamps when writing to the trace file.

```
* setdebug level=10 trace=1 options=ct fd
```

It is now possible to use a *class* of debug messages called tags to control the debug output of Bacula daemons.

all Display all debug messages

bvfs Display BVFS debug messages

sql Display SQL related debug messages

memory Display memory and poolmem allocation messages

scheduler Display scheduler related debug messages

```
* setdebug level=10 tags=bvfs,sql,memory
* setdebug level=10 tags=!bvfs

# bacula-dir -t -d 200,bvfs,sql
```

The tags option is composed of a list of tags. Tags are separated by "," or "+" or "-" or "!". To disable a specific tag, use "-" or "!" in front of the tag. Note that more tags are planned for future versions.

Table 54.1: Debug tag option table

Component	Tag	Debug Level	Comment
director	scheduler	100	information about job queue mangement
director	scheduler	20	information about resources in job queue
director	bvfs	10	information about bvfs
director	sql	15	information about bvfs queries
all	memory	40-60	information about smartalloc

54.11 Bacula Enterprise 6.6.0

54.11.1 Communication Line Compression

Bacula Enterprise version 6.6.0 and later now includes communication line compression. It is turned on by default, and if the two Bacula components (DIR, FD, SD, bconsole) are both version 6.6.0 or greater, communication line compression) will be enabled, by default. If for some reason, you do not want communication line compression, you may disable it with the following directive:

```
Comm Compression = no
```

This directive can appear in the following resources:

- `bacula-dir.conf`: Director resource
- `bacula-fd.conf`: Client (or FileDaemon) resource
- `bacula-sd.conf`: Storage resource

- bconsole.conf: Console resource
- bat.conf: Console resource

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is enabled. In the case that the compression is not effective, Bacula turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports None in the Job report.

54.11.2 Read Only Storage Devices

This version of Bacula allows you to define a Storage deamon device to be read-only. If the **Read Only** directive is specified and enabled, the drive can only be used for read operations. The **Read Only** directive can be defined in any bacula-sd.conf Device resource, and is most useful for reserving one or more drives for restores. An example is:

```
Read Only = yes
```

54.11.3 Catalog Performance Improvements

There is a new Bacula database format (schema) in this version of Bacula that eliminates the FileName table by placing the Filename into the File record of the File table. This substantiallly improves performance, particularly for large (1GB or greater) databases.

The update_xxx_catalog script will automatically update the Bacula database format, but you should realize that for very large databases (greater than 1GB), it may take some time, and there are several different options for doing the update:

1. Shudown the database and update it
2. Update the database while production jobs are running.

See the Bacula Systems White Paper "Migration-to-6.6" on this subject.

This database format change can provide very significant improvements in the speed of metadata insertion into the database, and in some cases (backup of large email servers) can significantly reduce the size of the database.

54.11.4 Plugin Restore Options

This version of Bacula permits user configuration of Plugins at restore time. For example, it is now possible to choose the datastore where your VMware image will be restored, or to choose pg_restore options directly. See specific Plugin whitepapers for more information about new restore options.

The restore options, if implemented in a plugin, will be presented to you during initiation of a restore either by command line or if available by a GUI such as BWeb. For examples of the command line interface and the GUI interface, please see below:

```
*run restore jobid=11766
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /tmp/regress/working/my-dir.restore.1.bsr
Where:           /tmp/regress/tmp/bacula-restores
...
Plugin Options:  *None*
OK to run? (yes/mod/no): mod
Parameters to modify:
     1: Level
...
    13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : vsphere: host=squeeze2
Plugin Restore Options
datastore:           *None*
restore_host:        *None*
new_hostname:        *None*
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
     1: datastore (Datastore to use for restore)
     2: restore_host (ESX host to use for restore)
     3: new_hostname (Restore host to specified name)
Select parameter to modify (1-3): 3
Please enter a value for new_hostname: test
Plugin Restore Options
datastore:           *None*
restore_host:        *None*
new_hostname:        test
Use above plugin configuration? (yes/mod/no): yes
```

Or via the BWeb restore interface (see Fig 54.22)

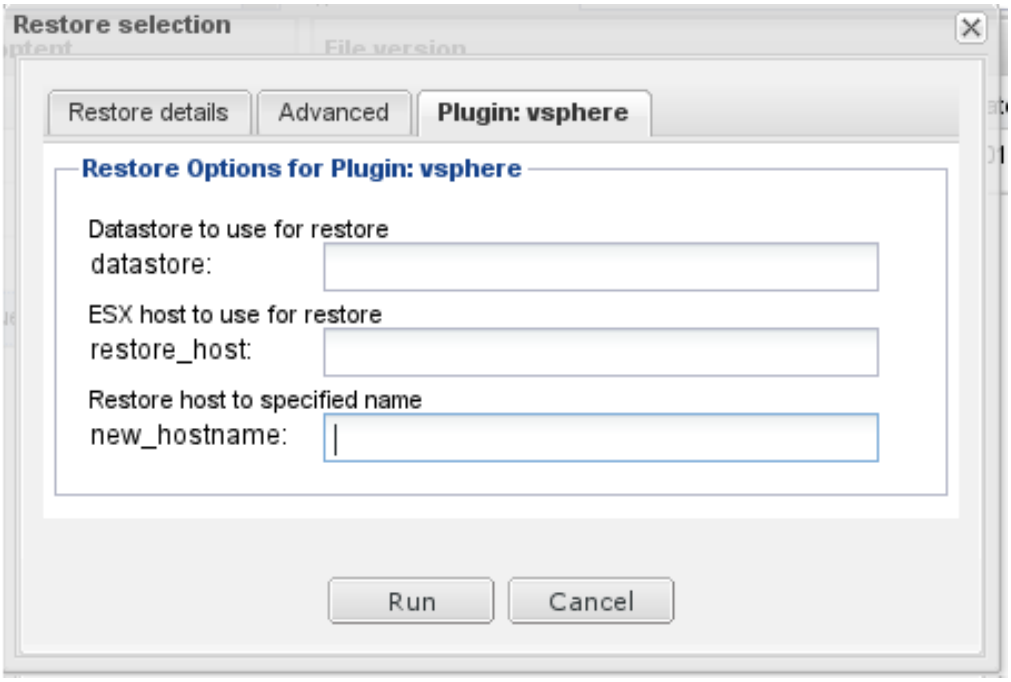


Figure 54.22: Choose datastore, ESXi or hostname at restore time

54.11.5 Alldrives Plugin Improvements

The `alldrives` plugin simplifies the FileSet creation of Windows Clients by automatically generating a FileSet which includes all local drives.

The alldrives plugin now accepts the snapshot option that generates snapshots for all local Windows drives, but without explicitly adding them to the FileSet. It may be combined with the VSS plugin. For example:

```
FileSet {
 ...
  Include {
    Plugin = "vss:/@MSSQL/"
    Plugin = "alldrives: snapshot"     # should be placed after vss plugin
  }
}
```

54.11.6 New Truncate Command

We have added a new truncate command to bconsole which will truncate a volume if the volume is purged, and if the volume is also marked **Action On Purge = Truncate**. This feature was originally added in Bacula version 5.0.1, but the mechanism for actually doing the truncate required the user to enter a complicated command such as:

```
purge volume action=truncate storage=File pool=Default
```

The above command is now simplified to be:

```
truncate storage=File pool=Default
```

54.12 Bacula Enterprise 6.4.x

The following features were added during the 6.4.x life cycle.

54.12.1 SAP Plugin

The Bacula Enterprise SAP Plugin is designed to implement the official SAP Backint interface to simplify the backup and restore procedure through your traditional SAP database tools. See SAP-Backint whitepaper for more information.

54.12.2 Oracle SBT Plugin

By default, the Oracle backup Manager, RMAN, sends all backups to an operating system specific directory on disk. You can also configure RMAN to make backups to media such as tape using the SBT module. Bacula will act as Media Manager, and the data will be transfered directly from RMAN to Bacula. See Oracle Plugin whitepaper for more information.

54.12.3 MySQL Plugin

The MySQL plugin is designed to simplify the backup and restore of your MySQL database, the backup administrator doesn't need to know about the internals of MySQL backup techniques or how to write complex scripts. This plugin will automatically backup essential information such as configurations and user definitions. The MySQL plugin supports both dump (with support for Incremental backup) and binary backup techniques. See the MySQL Plugin whitepaper for more information.

54.13 Bacula Enterprise 6.4.0

54.13.1 Deduplication Optimized Volumes

This version of Bacula includes a new alternative (or additional) volume format that optimizes the placement of files so that an underlying deduplicating filesystem such as ZFS can optimally deduplicate the backup data that is written by Bacula. These are called Deduplication Optimized Volumes or Aligned Volumes for short. The details of how to use this feature and its considerations are in the Bacula Systems Deduplication Optimized Volumes whitepaper.

54.13.2 Migration/Copy/VirtualFull Performance Enhancements

The Bacula Storage daemon now permits multiple jobs to simultaneously read from the same disk volume which gives substantial performance enhancements when running Migration, Copy, or VirtualFull jobs that read disk volumes. Our testing shows that when running multiple simultaneous jobs, the jobs can finish up to ten times faster with this version of Bacula. This is built-in to the Storage daemon, so it happens automatically and transparently.

54.13.3 VirtualFull Backup Consolidation Enhancements

By default Bacula selects jobs automatically for a VirtualFull backup. However, you may want to create the virtual backup based on a particular backup (point in time) that exists.

For example, if you have the following backup Jobs in your catalog:

```
+-------+---------+-------+----------+----------+-----------+
| JobId | Name    | Level | JobFiles | JobBytes | JobStatus |
+-------+---------+-------+----------+----------+-----------+
| 1     | Vbackup | F     | 1754     | 50118554 | T         |
| 2     | Vbackup | I     | 1        | 4        | T         |
| 3     | Vbackup | I     | 1        | 4        | T         |
| 4     | Vbackup | D     | 2        | 8        | T         |
| 5     | Vbackup | I     | 1        | 6        | T         |
| 6     | Vbackup | I     | 10       | 60       | T         |
| 7     | Vbackup | I     | 11       | 65       | T         |
| 8     | Save    | F     | 1758     | 50118564 | T         |
+-------+---------+-------+----------+----------+-----------+
```

and you want to consolidate only the first 3 jobs and create a virtual backup equivalent to Job 1 + Job 2 + Job 3, you will use `jobid=3` in the `run` command, then Bacula will select the previous Full backup, the previous Differential (if any) and all subsequent Incremental jobs.

```
run job=Vbackup jobid=3 level=VirtualFull
```

If you want to consolidate a specific job list, you must specify the exact list of jobs to merge in the `run` command line. For example, to consolidate the last Differential and all subsequent Incrementals, you will use `jobid=4,5,6,7` or `jobid=4-7` on the `run` command line. Because one of the Jobs in the list is a Differential backup, Bacula will set the new job level to Differential. If the list is composed of only Incremental jobs, the new job will have its level set to Incremental.

```
run job=Vbackup jobid=4-7 level=VirtualFull
```

When using this feature, Bacula will automatically discard jobs that are not related to the current Job. For example, specifying `jobid=7,8`, Bacula will discard JobId 8 because it is not part of the same backup Job.

We do not recommend it, but if you really want to consolidate jobs that have different names (so probably different clients, filesets, etc...), you must use `alljobid=` keyword instead of `jobid=`.

```
run job=Vbackup alljobid=1-3,6-8 level=VirtualFull
```

54.13.4 New Prune "Expired" Volume Command

In Bacula Enterprise 6.4, it is now possible to prune all volumes (from a pool, or globally) that are "expired". This option can be scheduled after or before the backup of the catalog and can be combined with the `Truncate On Purge` option. The `prune expired volme` command may be used instead of the `manual_prune.pl` script.

```
* prune expired volume

* prune expired volume pool=FullPool
```

To schedule this option automatically, it can be added to the Catalog backup job definition.

```
Job {
   Name = CatalogBackup
   ...
   RunScript {
     Console = "prune expired volume yes"
     RunsWhen = Before
   }
}
```

54.14 Bacula Enterprise 6.2.3

54.14.1 New Job Edit Codes %P %C

In various places such as RunScripts, you have now access to %P to get the current Bacula process ID (PID) and %C to know if the current job is a cloned job.

54.15 Bacula Enterprise 6.2.0

54.15.1 BWeb Bacula Configuration GUI

In Bacula Enterprise version 6.2, the BWeb Management Suite integrates a Bacula configuration GUI module which is designed to help you create and modify the Bacula configuration files such as `bacula-dir.conf`, `bacula-sd.conf`, `bacula-fd.conf` and `bconsole.conf`.

The BWeb Management Suite offers a number of Wizards which support the Administrator in his daily work. The wizards provide a step by step set of required actions that graphically guide the Administrator to perform quick and easy creation and modification of configuration files.

BWeb also provides diagnostic tools that enable the Administrator to check that the Catalog Database is well configured, and that BWeb is installed properly.

The new Online help mode displays automatic help text suggestions when the user searches data types.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

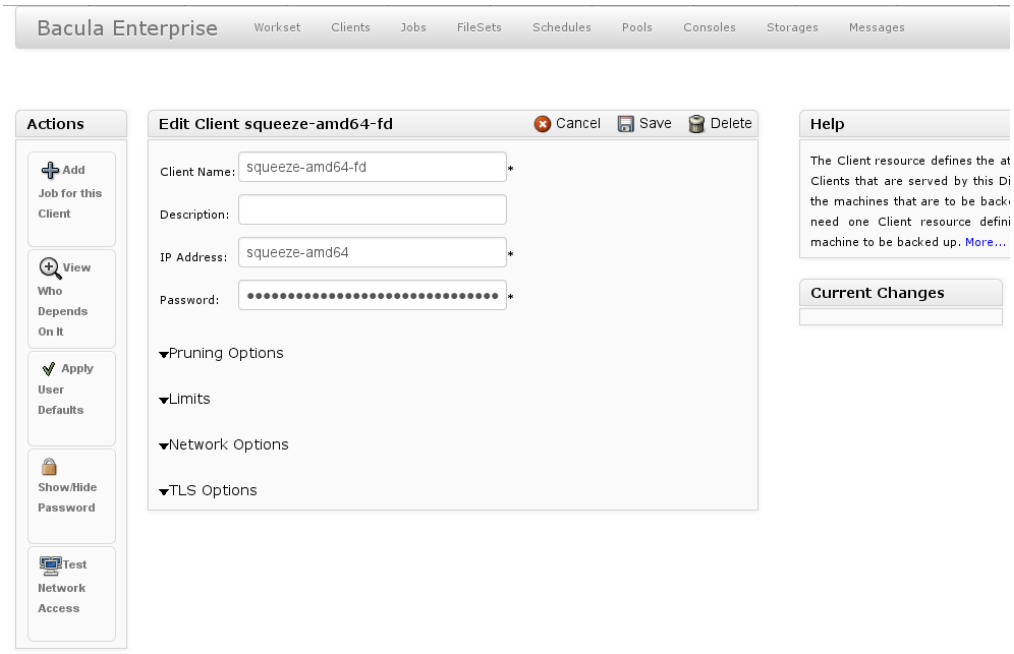


Figure 54.23: Configuration with BWeb Management Suite

54.15.2 Performance Improvements

Bacula Enterprise 6.2 has a number of new performance improvements:

- An improved way of storing Bacula Resources (as defined in the .conf files). This new handling permits much faster loading or reloading of the conf files, and permits larger numbers of resources.
- Improved performance when inserting large numbers of files in the DB catalog by breaking the insertion into smaller chunks, thus allowing better sharing when running multiple simultaneous jobs.
- Performance enhancements in BVFS concerning eliminating duplicate path records.
- Performance improvement when getting Pool records.
- Pruning performance enhancements.

54.15.3 Enhanced Status and Error Messages

We have enhanced the Storage daemon status output to be more readable. This is important when there are a large number of devices. In addition to formatting changes, it also includes more details on which devices are reading and writing.

A number of error messages have been enhanced to have more specific data on what went wrong.

If a file changes size while being backed up the old and new size are reported.

54.15.4 WinBMR 3

The Windows Bare Metal Recovery (BMR) plugin enables you to do safe, reliable Disaster Recovery with Bacula Enterprise Edition on Windows and allows you to get critical systems up and running again quickly. The Enterprise Edition Windows BMR is a toolkit that allows the

Administrator to perform the restore of a complete operating system to the same or similar hardware without actually going through the operating system's installation procedure.

The WinBMR 3 version is a major rewrite of the product that support all x86 Windows versions and technologies. Especially UEFI and secure boot systems. The WinBMR 3 File Daemon plugin is now part of the plugins included with the Bacula File Daemon package. The rescue CD or USB key is available separately.

54.15.5 Miscellaneous New Features

- Allow unlimited line lengths in .conf files (previously limited to 2000 characters).
- Allow `/dev/null` in ChangerCommand to indicated a Virtual Autochanger.
- Add a `--fileprune` option to the `manual_prune.pl` script.
- Add a `-m` option to `make_catalog_backup.pl` to do maintenance on the catalog.
- Safer code that cleans up the working directory when starting the daemons. It limits what files can be deleted, hence enhances security.
- Added a new `.ls` command in bconsole to permit browsing a client's filesystem.
- Fixed a number of bugs, includes some obscure seg faults, and a race condition that occurred infrequently when running Copy, Migration, or Virtual Full backups.
- Included a new vSphere library version, which will hopefully fix some of the more obscure bugs.
- Upgraded to a newer version of Qt4 for BAT. All indications are that this will improve BAT's stability on Windows machines.
- The Windows installers now detect and refuse to install on an OS that does not match the 32/64 bit value of the installer.

54.16 Bacula Enterprise 6.0.6

54.16.1 Incremental Accelerator Plugin for NetApp

The Incremental Accelerator for NetApp Plugin is designed to simplify the backup and restore procedure of your NetApp NAS hosting a huge number of files.

When using the NetApp HFC Plugin, Bacula Enterprise will query the NetApp device to get the list of all files modified since the last backup instead of having to walk through the entire filesystem. Once Bacula have the list of all files to back's up, it will use a standard network share (such as NFS or CIFS) to access files.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

54.16.2 `PostgreSQL` Plugin

The `PostgreSQL` plugin is designed to simplify the backup and restore procedure of your `PostgreSQL` cluster, the backup administrator doesn't need to learn about internals of `PostgreSQL` backup techniques or write complex scripts. The plugin will automatically take care for you to backup essential information such as configuration, users definition or tablespaces. The `PostgreSQL` plugin supports both dump and PITR backup techniques.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

54.16.3 Maximum Reload Requests

The new Director directive `Maximum Reload Requests` permits to configure the number of reload requests that can be done while jobs are running.

```
Director {
  Name = localhost-dir
  Maximum Reload Requests = 64
  ...
}
```

54.16.4 FD Storage Address

When the Director is behind a NAT, in a WAN area, to connect to the StorageDaemon, the Director uses an "external" ip address, and the FileDaemon should use an "internal" IP address to contact the StorageDaemon.

The normal way to handle this situation is to use a canonical name such as "storage-server" that will be resolved on the Director side as the WAN address and on the Client side as the LAN address. This is now possible to configure this parameter using the new directive **FDStorageAddress** in the Storage or Client resource.

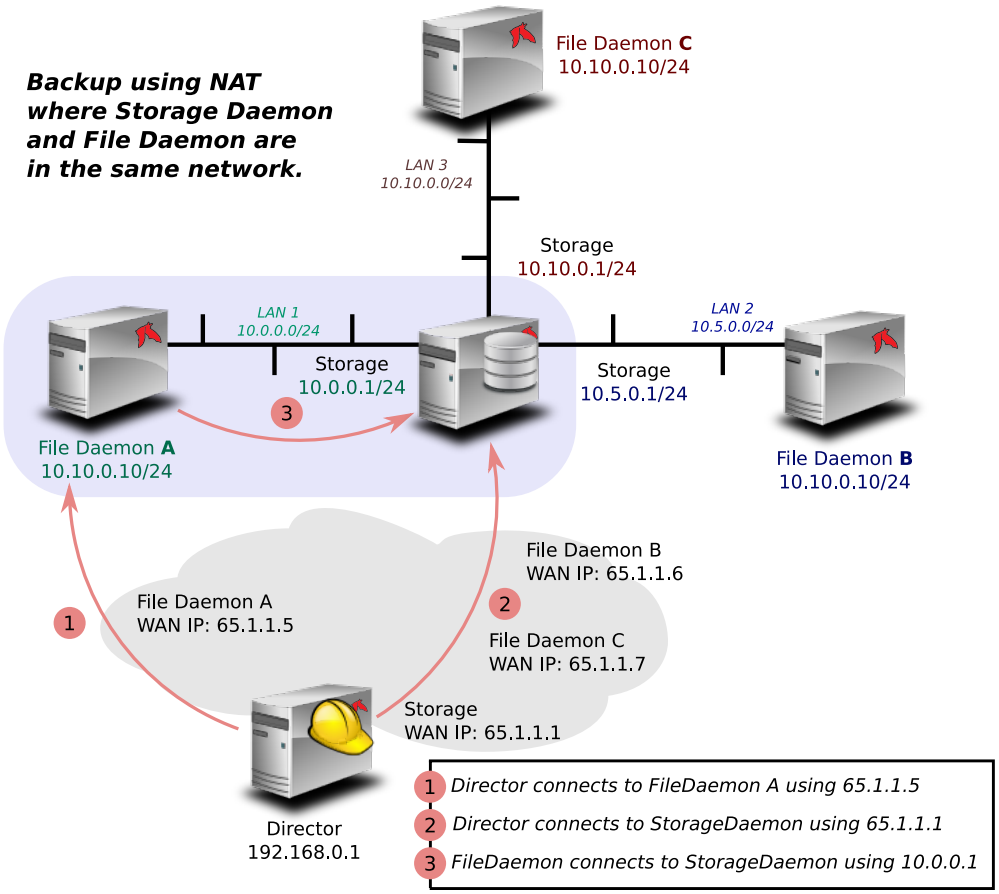


Figure 54.24: Backup Over WAN

```
Storage {
     Name = storage1
```

```
      Address = 65.1.1.1
      FD Storage Address = 10.0.0.1
      SD Port = 9103
      ...
}
```

```
Client {
     Name = client1
     Address = 65.1.1.2
     FD Storage Address = 10.0.0.1
     FD Port = 9102
     ...
}
```

Note that using the Client **FDStorageAddress** directive will not allow to use multiple Storage Daemon, all Backup or Restore requests will be sent to the specified **FDStorageAddress**.

54.16.5 Maximum Concurrent Read Jobs

This is a new directive that can be used in the `bacula-dir.conf` file in the Storage resource. The main purpose is to limit the number of concurrent Copy, Migration, and VirtualFull jobs so that they don't monopolize all the Storage drives causing a deadlock situation where all the drives are allocated for reading but none remain for writing. This deadlock situation can occur when running multiple simultaneous Copy, Migration, and VirtualFull jobs.

The default value is set to 0 (zero), which means there is no limit on the number of read jobs. Note, limiting the read jobs does not apply to Restore jobs, which are normally started by hand. A reasonable value for this directive is one half the number of drives that the Storage resource has rounded down. Doing so, will leave the same number of drives for writing and will generally avoid over committing drives and a deadlock.

54.17 Bacula Enterprise 6.0.4

54.17.1 VMWare vSphere VADP Plugin

The Bacula Enterprise vSphere plugin provides virtual machine bare metal recovery, while the backup at the guest level simplify data protection of critical applications.

The plugin integrates the VMware's CBT technology to ensure only blocks that have changed since the initial Full, and/or the last Incremental or Differential Backup are sent to the current Incremental or Differential backup stream to give you more efficient backups and reduced network load.

54.17.2 Oracle RMAN Plugin

The Bacula Enterprise Oracle Plugin is designed to simplify the backup and restore procedure of your Oracle Database instance, the backup administrator don't need to learn about internals of Oracle backup techniques or write complex scripts. The Bacula Enterprise Oracle plugin supports both dump and PITR with RMAN backup techniques.

54.18 Bacula Enterprise 6.0.2

To make Bacula function properly with multiple Autochanger definitions, in the Director's configuration, you must adapt your bacula-dir.conf **Storage** directives.

Each autochanger that you have defined in an **Autochanger** resource in the Storage daemon's bacula-sd.conf file, must have a corresponding **Autochanger** resource defined in the Director's bacula-dir.conf file. Normally you will already have a **Storage** resource that points to the Storage daemon's **Autochanger** resource. Thus you need only to change the name of the **Storage** resource to **Autochanger**. In addition the **Autochanger = yes** directive is not needed in the Director's **Autochanger** resource, since the resource name is **Autochanger**, the Director already knows that it represents an autochanger.

In addition to the above change (**Storage** to **Autochanger**), you must modify any additional **Storage** resources that correspond to devices that are part of the **Autochanger** device. Instead of the previous **Autochanger = yes** directive they should be modified to be **Autochanger = xxx** where you replace the **xxx** with the name of the Autochanger.

For example, in the bacula-dir.conf file:

```
Autochanger {             # New resource
  Name = Changer-1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO-Changer-1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 50
}

Storage {
  Name = Changer-1-Drive0
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive0
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

Storage {
  Name = Changer-1-Drive1
  Address = cibou.company.com
  SDPort = 9103
  Password = "xxxxxxxxxx"
  Device = LTO4_1_Drive1
  Media Type = LTO-4
  Maximum Concurrent Jobs = 5
  Autochanger = Changer-1  # New directive
}

...
```

Note that Storage resources **Changer-1-Drive0** and **Changer-1-Drive1** are not required since they make up part of an autochanger, and normally, Jobs refer only to the Autochanger resource. However, by referring to those Storage definitions in a Job, you will use only the indicated drive. This is not normally what you want to do, but it is very useful and often used for reserving a drive for restores. See the Storage daemon example .conf below and the use of **AutoSelect = no**.

So, in summary, the changes are:

- Change **Storage** to **Autochanger** in the LTO4 resource.
- Remove the **Autochanger = yes** from the **Autochanger** LTO4 resource.

- Change the **Autochanger = yes** in each of the **Storage** device that belong to the **Autochanger** to point to the **Autochanger** resource with for the example above the directive **Autochanger = LTO4**.

54.19 Bacula Enterprise 6.0.0

54.19.1 Incomplete Jobs

During a backup, if the Storage daemon experiences disconnection with the File daemon during backup (normally a comm line problem or possibly an FD failure), under conditions that the SD determines to be safe it will make the failed job as Incomplete rather than failed. This is done only if there is sufficient valid backup data that was written to the Volume. The advantage of an Incomplete job is that it can be restarted by the new bconsole `restart` command from the point where it left off rather than from the beginning of the jobs as is the case with a cancel.

54.19.2 The `stop` Command

Bacula has been enhanced to provide a `stop` command, very similar to the `cancel` command with the main difference that the Job that is stopped is marked as Incomplete so that it can be restarted later by the `restart` command where it left off (see below). The `stop` command with no arguments, will like the cancel command, prompt you with the list of running jobs allowing you to select one, which might look like the following:

```
*stop
Select Job:
     1: JobId=3 Job=Incremental.2012-03-26_12.04.26_07
     2: JobId=4 Job=Incremental.2012-03-26_12.04.30_08
     3: JobId=5 Job=Incremental.2012-03-26_12.04.36_09
Choose Job to stop (1-3): 2
2001 Job "Incremental.2012-03-26_12.04.30_08" marked to be stopped.
3000 JobId=4 Job="Incremental.2012-03-26_12.04.30_08" marked to be stopped.
```

54.19.3 The `restart` Command

The new `restart` command allows console users to restart a canceled, failed, or incomplete Job. For canceled and failed Jobs, the Job will restart from the beginning. For incomplete Jobs the Job will restart at the point that it was stopped either by a stop command or by some recoverable failure.

If you enter the `restart` command in bconsole, you will get the following prompts:

```
*restart
You have the following choices:
     1: Incomplete
     2: Canceled
     3: Failed
     4: All
Select termination code:  (1-4):
```

If you select the **All** option, you may see something like:

```
Select termination code:  (1-4): 4
+-------+-------------+---------------------+------+-------+----------+-----------+-----------+
| jobid | name        | starttime           | type | level | jobfiles | jobbytes  | jobstatus |
```

```
+-------+-------------+---------------------+------+-------+----------+-----------+-----------+
|     1 | Incremental | 2012-03-26 12:15:21 | B    | F     |        0 |         0 | A         |
|     2 | Incremental | 2012-03-26 12:18:14 | B    | F     |      350 | 4,013,397 | I         |
|     3 | Incremental | 2012-03-26 12:18:30 | B    | F     |        0 |         0 | A         |
|     4 | Incremental | 2012-03-26 12:18:38 | B    | F     |      331 | 3,548,058 | I         |
+-------+-------------+---------------------+------+-------+----------+-----------+-----------+
Enter the JobId list to select:
```

Then you may enter one or more JobIds to be restarted, which may take the form of a list of JobIds separated by commas, and/or JobId ranges such as **1-4**, which indicates you want to restart JobIds 1 through 4, inclusive.

54.19.4 Support for Exchange Incremental Backups

The Bacula Enterprise version 6.0 VSS plugin now supports Full and Incremental backups for Exchange. We strongly recommend that you do not attempt to run Differential jobs with Exchange as it is likely to produce a situation where restores will no longer select the correct jobs, and thus the Windows Exchange VSS writer will fail when applying log files. There is a Bacula Systems Enterprise white paper that provides the details of backup and restore of Exchange 2010 with the Bacula VSS plugin.

Restores can be done while Exchange is running, but you must first unmount (dismount in Microsoft terms) any database you wish to restore and explicitly mark them to permit a restore operation (see the white paper for details).

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

54.19.5 Support for MSSQL Block Level Backups

With the addition of block level backup support to the Bacula Enterprise VSS MSSQL component, you can now do Differential backups in addition to Full backups. Differential backups use Microsoft's partial block backup (a block differencing or deduplication that we call Delta). This partial block backup permits backing up only those blocks that have changed. Database restores can be made while the MSSQL server is running, but any databases selected for restore will be automatically taken offline by the MSSQL server during the restore process.

Incremental backups for MSSQL are not support by Microsoft. We strongly recommend that you not perform Incremental backups with MSSQL as they will probably produce a situation where restore will no longer work correctly.

We are currently working on producing a white paper that will give more details of backup and restore with MSSQL. One point to note is that during a restore, you will normally not want to restore the **master** database. You must exclude it from the backup selections that you have made or the restore will fail.

It is possible to restore the **master** database, but you must first shutdown the MSSQL server, then you must perform special recovery commands. Please see Microsoft documentation on how to restore the master database.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

54.19.6 Job Bandwidth Limitation

The new **Job Bandwidth Limitation** directive may be added to the File daemon's and/or Director's configuration to limit the bandwidth used by a Job on a Client. It can be set in the

File daemon's conf file for all Jobs run in that File daemon, or it can be set for each Job in the Director's conf file. The speed is always specified in bytes per second.

For example:

```
FileDaemon {
  Name = localhost-fd
  Working Directory = /some/path
  Pid Directory = /some/path
  ...
  Maximum Bandwidth Per Job = 5Mb/s
}
```

The above example would cause any jobs running with the FileDaemon to not exceed 5 megabytes per second of throughput when sending data to the Storage Daemon. Note, the speed is always specified in bytes per second (not in bits per second), and the case (upper/lower) of the specification characters is ignored (i.e. 1MB/s = 1Mb/s).

You may specify the following speed parameter modifiers: k/s (1,024 bytes per second), kb/s (1,000 bytes per second), m/s (1,048,576 bytes per second), or mb/s (1,000,000 bytes per second).

For example:

```
Job {
  Name = locahost-data
  FileSet = FS_localhost
  Accurate = yes
  ...
  Maximum Bandwidth = 5Mb/s
  ...
}
```

The above example would cause Job `localhost-data` to not exceed 5MB/s of throughput when sending data from the File daemon to the Storage daemon.

A new console command `setbandwidth` permits to set dynamically the maximum throughput of a running Job or for future jobs of a Client.

```
* setbandwidth limit=1000 jobid=10
```

Number of bytes can be expressed using modifiers mentioned above (k/s, kb/s, m/s or mb/s).

This project was funded by Bacula Systems and is available in Bacula Enterprise Edition.

54.19.7 Incremental/Differential Block Level Difference Backup

The new `delta` Plugin is able to compute and apply signature-based file differences. It can be used to backup only changes in a big binary file like Outlook PST, VirtualBox/VMware images or database files.

It supports both Incremental and Differential backups and stores signatures database in the File Daemon working directory. This plugin is available on all platform including Windows 32 and 64bit.

Accurate option should be turned on in the Job resource.

```
Job {
 Accurate = yes
```

```
 FileSet = DeltaFS
 ...
}

FileSet {
 Name = DeltaFS
 ...
 Include {
   # Specify one file
   Plugin = "delta:/home/eric/.VirtualBox/HardDisks/lenny-i386.vdi"
 }
}

FileSet {
 Name = DeltaFS-Include
 ...
 Include {
   Options {
      Compression = GZIP1
      Signature = MD5
      Plugin = delta
   }
   # Use the Options{} filtering and options
   File = /home/user/.VirtualBox
 }
}
```

Please contact Bacula Systems support to get Delta Plugin specific documentation.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition.

54.19.8 SAN Shared Tape Storage Plugin

The problem with backing up multiple servers at the same time to the same tape library (or autoloader) is that if both servers access the same tape drive same time, you will very likely get data corruption. This is where the Bacula Systems shared tape storage plugin comes into play. The plugin ensures that only one server at a time can connect to each device (tape drive) by using the SPC-3 SCSI reservation protocol. Please contact Bacula Systems support to get SAN Shared Storage Plugin specific documentation.

This project was funded by Bacula Systems and is available with Bacula Enterprise Edition.

54.19.9 Advanced Autochanger Usage

The new `Shared Storage` Director's directive is a Bacula Enterprise feature that allows you to share volumes between different Storage resources. This directive should be used **only** if all `Media Type` are correctly set across all Devices.

The `Shared Storage` directive should be used when using the SAN Shared Storage plugin or when accessing from the Director Storage resources directly to Devices of an Autochanger.

When sharing volumes between different Storage resources, you will need also to use the `reset-storageid` script before using the `update slots` command. This script can be scheduled once a day in an Admin job.

```
$ /opt/bacula/scripts/reset-storageid MediaType StorageName
$ bconsole
* update slots storage=StorageName drive=0
```

Please contact Bacula Systems support to get help on this advanced configuration.

This project was funded by Bacula Systems and is available with Bacula Enterprise Edition.

The reset-storageid procedure is no longer required when using the appropriate Autochanger configuration in the Director configuration side.

54.19.10 Enhancement of the NDMP Plugin

The previous NDMP Plugin 4.0 was fully supporting only the NetApp hardware, the new NDMP Plugin should now be able to support all NAS vendors with the `volume_format` plugin command option.

On some NDMP devices such as Celera or Blueray, the administrator can use arbitrary volume structure name, ex:

```
/dev/volume_home
/rootvolume/volume_tmp
/VG/volume_var
```

The NDMP plugin should be aware of the structure organization in order to detect if the administrator wants to restore in a new volume (`where=/dev/vol_tmp`) or inside a subdirectory of the targeted volume (`where=/tmp`).

```
FileSet {
 Name = NDMPFS
 ...
 Include {
   Plugin = "ndmp:host=nasbox user=root pass=root file=/dev/vol1 volume_format=/dev/"
 }
}
```

Please contact Bacula Systems support to get NDMP Plugin specific documentation.

This project was funded by Bacula Systems and is available with the Bacula Enterprise Edition

54.19.11 Always Backup a File

When the Accurate mode is turned on, you can decide to always backup a file by using then new **A** Accurate option in your FileSet. For example:

```
Job {
   Name = ...
   FileSet = FS_Example
   Accurate = yes
   ...
}

FileSet {
 Name = FS_Example
 Include {
   Options {
     Accurate = A
   }
   File = /file
   File = /file2
 }
 ...
}
```

This project was funded by Bacula Systems based on an idea of James Harper and is available with the Bacula Enterprise Edition.

54.19.12 Setting Accurate Mode at Runtime

You are now able to specify the Accurate mode on the run command and in the Schedule resource.

```
* run accurate=yes job=Test
```

```
Schedule {
  Name = WeeklyCycle
  Run = Full 1st sun at 23:05
  Run = Differential accurate=yes 2nd-5th sun at 23:05
  Run = Incremental  accurate=no  mon-sat at 23:05
}
```

It can allow you to save memory and and CPU resources on the catalog server in some cases.

These advanced tuning options are available with the Bacula Enterprise Edition.

54.19.13 Additions to RunScript variables

You can have access to JobBytes, JobFiles and Director name using %b, %F and %D in your runscript command. The Client address is now available through %h.

```
RunAfterJob = "/bin/echo Job=%j JobBytes=%b JobFiles=%F ClientAddress=%h Dir=%D"
```

54.19.14 LZO Compression

LZO compression was added in the Unix File Daemon. From the user point of view, it works like the GZIP compression (just replace **compression=GZIP** with **compression=LZO**).

For example:

```
Include {
   Options { compression=LZO }
   File = /home
   File = /data
}
```

LZO provides much faster compression and decompression speed but lower compression ratio than GZIP. It is a good option when you backup to disk. For tape, the built-in compression may be a better option.

LZO is a good alternative for GZIP1 when you don't want to slow down your backup. On a modern CPU it should be able to run almost as fast as:

- your client can read data from disk. Unless you have very fast disks like SSD or large/fast RAID array.
- the data transfers between the file daemon and the storage daemon even on a 1Gb/s link.

Note that bacula only use one compression level LZO1X-1.

The code for this feature was contributed by Laurent Papier.

54.19.15 New Tray Monitor

Since the old integrated Windows tray monitor doesn't work with recent Windows versions, we have written a new Qt Tray Monitor that is available for both Linux and Windows. In addition to all the previous features, this new version allows you to run Backups from the tray monitor menu.

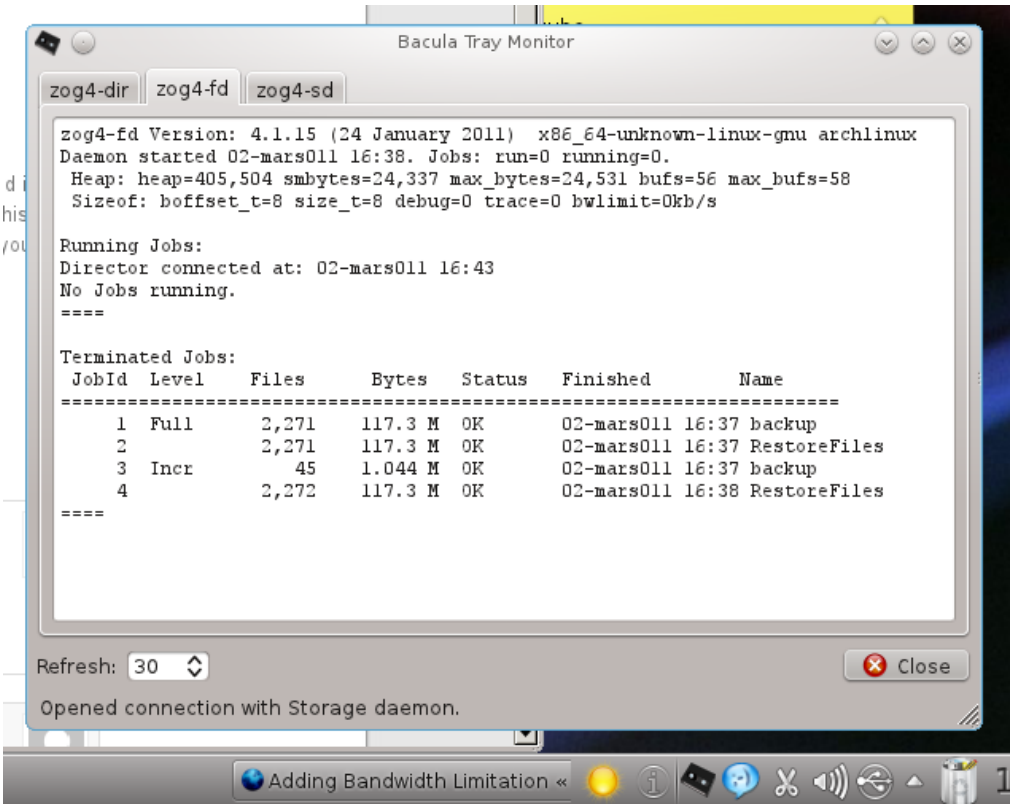


Figure 54.25: New tray monitor

To be able to run a job from the tray monitor, you need to allow specific commands in the Director monitor console:

```
Console {
    Name = win2003-mon
    Password = "xxx"
    CommandACL = status, .clients, .jobs, .pools, .storage, .filesets, .messages, run
    ClientACL = *all*               # you can restrict to a specific host
    CatalogACL = *all*
    JobACL = *all*
    StorageACL = *all*
    ScheduleACL = *all*
    PoolACL = *all*
    FileSetACL = *all*
    WhereACL = *all*
}
```

This project was funded by Bacula Systems and is available with Bacula Enterprise Edition and Bacula Community Edition.

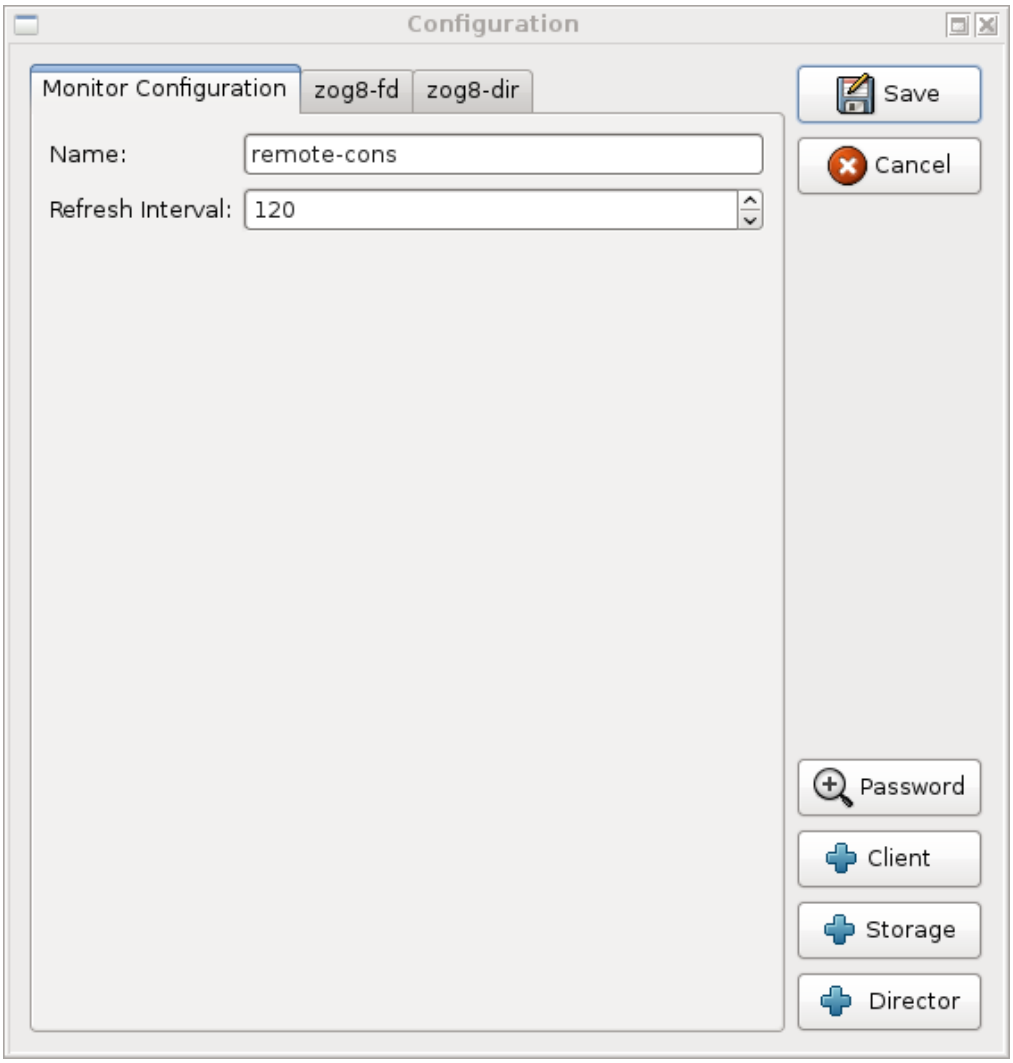


Figure 54.26: Run a Job through the new tray monitor

54.19.16 Purge Migration Job

The new **Purge Migration Job** directive may be added to the Migration Job definition in the `Director`'s configuration file. When it is enabled the Job that was migrated during a migration will be purged at the end of the migration job.

For example:

```
Job {
  Name = "migrate-job"
  Type = Migrate
  Level = Full
  Client = localhost-fd
  FileSet = "Full Set"
  Messages = Standard
  Storage = DiskChanger
  Pool = Default
  Selection Type = Job
  Selection Pattern = ".*Save"
...
  Purge Migration Job = yes
}
```

This project was submitted by Dunlap Blake; testing and documentation was funded by Bacula Systems.

54.19.17 Changes in the Pruning Algorithm

We rewrote the job pruning algorithm in this version. Previously, in some users reported that the pruning process at the end of jobs was very long. It should not be longer the case. Now, Bacula won't prune automatically a Job if this particular Job is needed to restore data. Example:

```
JobId: 1  Level: Full
JobId: 2  Level: Incremental
JobId: 3  Level: Incremental
JobId: 4  Level: Differential
.. Other incrementals up to now
```

In this example, if the Job Retention defined in the Pool or in the Client resource causes that Jobs with Jobid in 1,2,3,4 can be pruned, Bacula will detect that JobId 1 and 4 are essential to restore data at the current state and will prune only JobId 2 and 3.

`Important`, this change affect only the automatic pruning step after a Job and the `prune jobs` Bacula **Console** command. If a volume expires after the **VolumeRetention** period, important jobs can be pruned.

54.19.18 Ability to Verify any specified Job

You now have the ability to tell Bacula which Job should verify instead of automatically verify just the last one.

This feature can be used with VolumeToCatalog, DiskToCatalog and Catalog level.

To verify a given job, just specify the Job jobid in argument when starting the job.

```
*run job=VerifyVolume jobid=1 level=VolumeToCatalog
Run Verify job
JobName:     VerifyVolume
```

```
Level:       VolumeToCatalog
Client:      127.0.0.1-fd
FileSet:     Full Set
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  VerifyVol.2010-09-08_14.17.17_03
Verify List: /tmp/regress/working/VerifyVol.bsr
When:        2010-09-08 14:17:31
Priority:    10
OK to run? (yes/mod/no):
```

This project was funded by Bacula Systems and is available with Bacula Enterprise Edition and Community Edition.

Chapter 55

The Bootstrap File

The information in this chapter is provided so that you may either create your own bootstrap files, or so that you can edit a bootstrap file produced by **Bacula**. However, normally the bootstrap file will be automatically created for you during the **Bacula Console** chapter (chapter 1 page 1) of the Bacula Community Edition Console manual, or by using a Write Bootstrap record in your Backup Jobs, and thus you will never need to know the details of this file.

The **bootstrap** file contains ASCII information that permits precise specification of what files should be restored, what volume they are on, and where they are on the volume. It is a relatively compact form of specifying the information, is human readable, and can be edited with any text editor.

55.1 Bootstrap File Format

The general format of a **bootstrap** file is:

<**keyword**>= <**value**>

Where each **keyword** and the **value** specify which files to restore. More precisely the **keyword** and their **values** serve to limit which files will be restored and thus act as a filter. The absence of a keyword means that all records will be accepted.

Blank lines and lines beginning with a pound sign (#) in the bootstrap file are ignored.

There are keywords which permit filtering by Volume, Client, Job, FileIndex, Session Id, Session Time, ...

The more keywords that are specified, the more selective the specification of which files to restore will be. In fact, each keyword is **AND**ed with other keywords that may be present.

For example,

```
Volume = Test-001
VolSessionId = 1
VolSessionTime = 108927638
```

directs the Storage daemon (or the bextract program) to restore only those files on Volume Test-001 **AND** having VolumeSessionId equal to one **AND** having VolumeSession time equal to 108927638.

The full set of permitted keywords presented in the order in which they are matched against the Volume records are:

Volume The value field specifies what Volume the following commands apply to. Each Volume specification becomes the current Volume, to which all the following commands apply until a new current Volume (if any) is specified. If the Volume name contains spaces, it should be enclosed in quotes. At lease one Volume specification is required.

Count The value is the total number of files that will be restored for this Volume. This allows the Storage daemon to know when to stop reading the Volume. This value is optional.

VolFile The value is a file number, a list of file numbers, or a range of file numbers to match on the current Volume. The file number represents the physical file on the Volume where the data is stored. For a tape volume, this record is used to position to the correct starting file, and once the tape is past the last specified file, reading will stop.

VolBlock The value is a block number, a list of block numbers, or a range of block numbers to match on the current Volume. The block number represents the physical block within the file on the Volume where the data is stored.

VolSessionTime The value specifies a Volume Session Time to be matched from the current volume.

VolSessionId The value specifies a VolSessionId, a list of volume session ids, or a range of volume session ids to be matched from the current Volume. Each VolSessionId and VolSessionTime pair corresponds to a unique Job that is backed up on the Volume.

JobId The value specifies a JobId, list of JobIds, or range of JobIds to be selected from the current Volume. Note, the JobId may not be unique if you have multiple Directors, or if you have reinitialized your database. The JobId filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

Job The value specifies a Job name or list of Job names to be matched on the current Volume. The Job corresponds to a unique VolSessionId and VolSessionTime pair. However, the Job is perhaps a bit more readable by humans. Standard regular expressions (wildcards) may be used to match Job names. The Job filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

Client The value specifies a Client name or list of Clients to will be matched on the current Volume. Standard regular expressions (wildcards) may be used to match Client names. The Client filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

FileIndex The value specifies a FileIndex, list of FileIndexes, or range of FileIndexes to be selected from the current Volume. Each file (data) stored on a Volume within a Session has a unique FileIndex. For each Session, the first file written is assigned FileIndex equal to one and incremented for each file backed up.

This for a given Volume, the triple VolSessionId, VolSessionTime, and FileIndex uniquely identifies a file stored on the Volume. Multiple copies of the same file may be stored on the same Volume, but for each file, the triple VolSessionId, VolSessionTime, and FileIndex will be unique. This triple is stored in the Catalog database for each file.

To restore a particular file, this value (or a range of FileIndexes) is required.

FileRegex The value is a regular expression. When specified, only matching filenames will be restored.

```
FileRegex=^/etc/passwd(.old)?
```

Slot The value specifies the autochanger slot. There may be only a single **Slot** specification for each Volume.

Stream The value specifies a Stream, a list of Streams, or a range of Streams to be selected from the current Volume. Unless you really know what you are doing (the internals of **Bacula**), you should avoid this specification. This value is optional and not used by Bacula to restore files.

*JobType Not yet implemented.

*JobLevel Not yet implemented.

The **Volume** record is a bit special in that it must be the first record. The other keyword records may appear in any order and any number following a Volume record.

Multiple Volume records may be specified in the same bootstrap file, but each one starts a new set of filter criteria for the Volume.

In processing the bootstrap file within the current Volume, each filter specified by a keyword is **AND**ed with the next. Thus,

```
Volume = Test-01
Client = "My machine"
FileIndex = 1
```

will match records on Volume **Test-01 AND** Client records for **My machine AND** FileIndex equal to **one**.

Multiple occurrences of the same record are **OR**ed together. Thus,

```
Volume = Test-01
Client = "My machine"
Client = "Backup machine"
FileIndex = 1
```

will match records on Volume **Test-01 AND** (Client records for **My machine OR Backup machine**) **AND** FileIndex equal to **one**.

For integer values, you may supply a range or a list, and for all other values except Volumes, you may specify a list. A list is equivalent to multiple records of the same keyword. For example,

```
Volume = Test-01
Client = "My machine", "Backup machine"
FileIndex = 1-20, 35
```

will match records on Volume **Test-01 AND (**Client records for **My machine OR Backup machine) AND (**FileIndex 1 **OR** 2 **OR** 3 . . . **OR** 20 **OR** 35**)**.

As previously mentioned above, there may be multiple Volume records in the same bootstrap file. Each new Volume definition begins a new set of filter conditions that apply to that Volume and will be **OR**ed with any other Volume definitions.

As an example, suppose we query for the current set of tapes to restore all files on Client **Rufus** using the `query` command in the console program:

```
Using default Catalog name=MySQL DB=bacula
*query
Available queries:
     1: List Job totals:
     2: List where a file is saved:
     3: List where the most recent copies of a file are saved:
     4: List total files/bytes by Job:
     5: List total files/bytes by Volume:
     6: List last 10 Full Backups for a Client:
     7: List Volumes used by selected JobId:
     8: List Volumes to Restore All Files:
Choose a query (1-8): 8
Enter Client Name: Rufus
+-------+-----------------+------------+-----------+----------+------------+
| JobId | StartTime       | VolumeName | StartFile | VolSesId | VolSesTime |
+-------+-----------------+------------+-----------+----------+------------+
```

```
| 154   | 2002-05-30 12:08 | test-02    | 0         | 1         | 1022753312 |
| 202   | 2002-06-15 10:16 | test-02    | 0         | 2         | 1024128917 |
| 203   | 2002-06-15 11:12 | test-02    | 3         | 1         | 1024132350 |
| 204   | 2002-06-18 08:11 | test-02    | 4         | 1         | 1024380678 |
+-------+------------------+------------+-----------+-----------+------------+
```

The output shows us that there are four Jobs that must be restored. The first one is a Full backup, and the following three are all Incremental backups.

The following bootstrap file will restore those files:

```
Volume=test-02
VolSessionId=1
VolSessionTime=1022753312
Volume=test-02
VolSessionId=2
VolSessionTime=1024128917
Volume=test-02
VolSessionId=1
VolSessionTime=1024132350
Volume=test-02
VolSessionId=1
VolSessionTime=1024380678
```

As a final example, assume that the initial Full save spanned two Volumes. The output from `query` might look like:

```
+-------+------------------+------------+-----------+-----------+------------+
| JobId | StartTime        | VolumeName | StartFile | VolSesId | VolSesTime |
+-------+------------------+------------+-----------+-----------+------------+
| 242   | 2002-06-25 16:50 | File0003   | 0         | 1         | 1025016612 |
| 242   | 2002-06-25 16:50 | File0004   | 0         | 1         | 1025016612 |
| 243   | 2002-06-25 16:52 | File0005   | 0         | 2         | 1025016612 |
| 246   | 2002-06-25 19:19 | File0006   | 0         | 2         | 1025025494 |
+-------+------------------+------------+-----------+-----------+------------+
```

and the following bootstrap file would restore those files:

```
Volume=File0003
VolSessionId=1
VolSessionTime=1025016612
Volume=File0004
VolSessionId=1
VolSessionTime=1025016612
Volume=File0005
VolSessionId=2
VolSessionTime=1025016612
Volume=File0006
VolSessionId=2
VolSessionTime=1025025494
```

55.2 Automatic Generation of Bootstrap Files

One thing that is probably worth knowing: the bootstrap files that are generated automatically at the end of the job are not as optimized as those generated by the `restore` command. This is because during Incremental and Differential jobs, the records pertaining to the files written for the Job are appended to the end of the bootstrap file. As consequence, all the files saved to an Incremental or Differential job will be restored first by the Full save, then by any Incremental or Differential saves.

When the bootstrap file is generated for the `restore` command, only one copy (the most recent) of each file is restored.

So if you have spare cycles on your machine, you could optimize the bootstrap files by doing the following:

```
./bconsole
restore client=xxx select all
done
no
quit
Backup bootstrap file.
```

The above will not work if you have multiple FileSets because that will be an extra prompt. However, the `restore client=xxx select all` builds the in-memory tree, selecting everything and creates the bootstrap file.

The **no** answers the **Do you want to run this (yes/mod/no)** question.

55.3 Bootstrap for bscan

If you have a very large number of Volumes to scan with `bscan`, you may exceed the command line limit (511 characters). I that case, you can create a simple bootstrap file that consists of only the volume names. An example might be:

```
Volume="Vol001"
Volume="Vol002"
Volume="Vol003"
Volume="Vol004"
Volume="Vol005"
```

55.4 A Final Bootstrap Example

If you want to extract or copy a single Job, you can do it by selecting by JobId (code not tested) or better yet, if you know the VolSessionTime and the VolSessionId (printed on Job report and in Catalog), specifying this is by far the best. Using the VolSessionTime and VolSessionId is the way Bacula does restores. A bsr file might look like the following:

```
Volume="Vol001"
VolSessionId=10
VolSessionTime=1080847820
```

If you know how many files are backed up (on the job report), you can enormously speed up the selection by adding (let's assume there are 157 files):

```
FileIndex=1-157
Count=157
```

Finally, if you know the File number where the Job starts, you can also cause `bcopy` to forward space to the right file without reading every record:

```
VolFile=20
```

There is nothing magic or complicated about a BSR file. Parsing it and properly applying it within Bacula *is* magic, but you don't need to worry about that.

If you want to see a **real** bsr file, simply fire up the `restore` command in the console program, select something, then answer no when it prompts to run the job. Then look at the file `restore.bsr` in your working directory.

Appendices

Appendix A

Job status

A.1 Job levels

The table A.1 describes the several levels for a job.

Table A.1: Job levels

Level	Description
Backup levels	
F	Full backup: Every files
I	Incremental: Files modified since last backup
D	Differential: Files modified since last full backup
S	Since: Not used
f	Virtual full backup
Verification levels	
C	Verify from Catalog
V	Verify: Init database
O	Verify volume to Catalog entries
d	Verify disk attributes to Catalog
A	Verify data on volume
Others	
B	Base level job
—	None: for Restore and Admin

A.2 Job types

The table A.2 describes the several type for a job.

Table A.2: Job types

Type	Description
B	Backup Job
V	Verify Job
R	Restore Job

Continues on the following page

[]Cont.

Type	Description
D	Admin job
C	Copy of a Job
c	Copy Job
M	A previous backup job that was migrated
g	Migration Job
A	Archive Job
S	Scan Job
U	Console program
I	Internal system “job”

A.3 Job status

The table A.3 describes the several status for a job.

Table A.3: Job Status

Status	Description
A	Job canceled by user
B	Job blocked
C	Job created but not yet running
D	Verify differences
E	Job terminated in error
F	Job waiting on File daemon
I	Incomplete Job
L	Committing data (last despool)
M	Job waiting for Mount
R	Job running
S	Job waiting on the Storage daemon
T	Job terminated normally
W	Job terminated normally with warnings
a	SD despooling attributes
c	Waiting for Client resource
d	Waiting for maximum jobs
e	Non-fatal error
f	Fatal error
i	Doing batch insert file records
j	Waiting for job resource
l	Doing data despooling
m	Waiting for new media
p	Waiting for higher priority jobs to finish
q	Queued waiting for device
s	Waiting for storage resource
t	Waiting for start time

Appendix B

Bacula Copyright, Trademark, and Licenses

There are a number of different licenses that are used in Bacula. If you have a printed copy of this manual, the details of each of the licenses referred to in this chapter can be found in the online version of the manual at http://www.baculasystems.com.

B.1 Bacula Systems

This manual and the associated code is the Enterprise version of Bacula. It is not open source or licensed under the AGPLv3 as is the Bacula community source code. Bacula Enterprise software is proprietary.

It is possible that some of it may still be marked as copyright Free Software Foundation Europe e.V. by error, but due to the dual copyright, for these purposes it should be considered copyright Bacula Systems SA with All Rights Reserved. In addition, there are parts which are copyright Bacula Systems SA, and parts which contain Bacula Systems' trademarks.

All source code listed under Copyright Bacula Systems SA as well as all code falling under the dual license (see above) is available only to users with a valid Bacula Systems subscription agreement, and may not be redistributed without explicit written permission from Bacula Systems SA.

Portions may be copyrighted by other people. These files are released under different licenses which are compatible with the Bacula Enterprise license.

B.2 Trademarks

The name Bacula Systems(R) and associated logos as well as other trademarks are trademarks of Bacula Systems SA. Absent a written agreement or other express permission, Bacula Systems SA does not allow third parties to use its trademarks. Absent such a written trademark license from Bacula Systems, a third party does not have the right to distribute anything (code, binaries, ...) containing Bacula Systems trademarks.

Bacula® is a registered trademark of Kern Sibbald.

B.3 LGPL

Some of the Bacula library source code is released under the GNU Lesser General Public License. This permits third parties to use these parts of our code in their proprietary programs to interface to Bacula.

B.4 Public Domain

Some of the Bacula code, or code that Bacula references, has been released to the public domain. E.g. md5.c, SQLite.

B.5 Fiduciary License Agreement

Developers who have contributed significant changes to the Bacula code should have signed a Fiduciary License Agreement (FLA), which guarantees them the right to use the code they have developed, and also ensures that the Free Software Foundation Europe (and thus the Bacula project and Bacula Systems) has the rights to the code. This Fiduciary License Agreement is found on the Bacula web site at:

http://www.bacula.org/en/FLA-bacula.en.pdf

and if you are submitting code, you should fill it out then sent to:

Kern Sibbald
Cotes-de-Montmoiret 9
1012 Lausanne
Switzerland

When you send in such a complete document, please notify me: kern at sibbald dot com.

B.6 Disclaimer

NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Appendix C

Thanks

I thank everyone who has helped this project. Unfortunately, I cannot thank everyone (bad memory). However, the AUTHORS file in the main source code directory should include the names of all persons who have contributed to the Bacula project. Just the same, I would like to include thanks below to special contributors as well as to the major contributors to the current release.

Thanks to Richard Stallman for starting the Free Software movement and for bringing us gcc and all the other GNU tools as well as the GPL license.

Thanks to Linus Torvalds for bringing us Linux.

Thanks to all the Free Software programmers. Without being able to peek at your code, and in some cases, take parts of it, this project would have been much more difficult.

Thanks to John Walker for suggesting this project, giving it a name, contributing software he has written, and for his programming efforts on Bacula as well as having acted as a constant sounding board and source of ideas.

Thanks to the apcupsd project where I started my Free Software efforts, and from which I was able to borrow some ideas and code that I had written.

Special thanks to D. Scott Barninger for writing the bacula RPM spec file, building all the RPM files and loading them onto Source Forge. This has been a tremendous help.

Many thanks to Karl Cunningham for converting the manual from html format to LaTeX. It was a major effort flawlessly done that will benefit the Bacula users for many years to come. Thanks Karl.

Thanks to Dan Langille for the **incredible** amount of testing he did on FreeBSD. His perseverance is truly remarkable. Thanks also for the many contributions he has made to improve Bacula (pthreads patch for FreeBSD, improved start/stop script and addition of Bacula userid and group, stunnel, ...), his continuing support of Bacula users. He also wrote the PostgreSQL driver for Bacula and has been a big help in correcting the SQL.

Thanks to multiple other Bacula Packagers who make and release packages for different platforms for Bacula.

Thanks to Christopher Hull for developing the native Windows Bacula emulation code and for contributing it to the Bacula project.

Thanks to Robert Nelson for bringing our Windows implementation up to par with all the same features that exist in the Unix/Linux versions.

Thanks to Thorsten Engel for his excellent knowledge of Win32 systems, and for making the Windows File daemon Unicode compatible, as well as making the Windows File daemon interface to Microsoft's Volume Shadow Copy (VSS). These two are big pluses for Bacula!

Thanks to Landon Fuller for writing both the communications and the data encryption code for Bacula.

Thanks to Arno Lehmann for his excellent and infatigable help and advice to users.

Thanks to all the Bacula users, especially those of you who have contributed ideas, bug reports, patches, and new features.

Bacula can be enabled with data encryption and/or communications encryption. If this is the case, you will be including OpenSSL code that that contains cryptographic software written by Eric Young[1] and also software written by Tim Hudson[2] .

The BAT (Bacula Administration Tool) graphs are based in part on the work of the Qwt project[3].

The original variable expansion code used in the LabelFormat comes from the Open Source Software Project[4]. It has been adapted and extended for use in Bacula. This code is now deprecated.

There have been numerous people over the years who have contributed ideas, code, and help to the Bacula project. The file AUTHORS in the main source release file contains a list of contributors. For all those who I have left out, please send me a reminder, and in any case, thanks for your contribution.

Thanks to the Free Software Foundation Europe e.V. for assuming the responsibilities of protecting the Bacula copyright.

Copyrights and Trademarks

Certain words and/or products are Copyrighted or Trademarked such as Windows (by Microsoft). Since they are numerous, and we are not necessarily aware of the details of each, we don't try to list them here. However, we acknowledge all such Copyrights and Trademarks, and if any copyright or trademark holder wishes a specific acknowledgment, notify us, and we will be happy to add it where appropriate.

[1] mailto:eay@cryptsoft.com
[2] mailto:tjh@cryptsoft.com
[3] http://qwt.sf.net
[4] www.ossp.org

C.1 Bacula Bugs

Well fortunately there are not too many bugs, but thanks to Dan Langille, we have a bugs database[5] where bugs are reported. Generally, when a bug is fixed, a patch for the currently released version will be attached to the bug report.

The directory `patches` in the current SVN always contains a list of the patches that have been created for the previously released version of Bacula. In addition, the file `patches-version-number` in the `patches` directory contains a summary of each of the patches.

A "raw" list of the current task list and known issues can be found in `kernstodo` in the main Bacula source directory.

[5] http://bugs.bacula.org

Appendix D

Acronyms

ACL Access Control List
AES Advanced Encryption Standard
ANSI American National Standards Institute
API Application Programming Interface
ASCII American Standard Code for Information Interchange
ASN.1 Abstract Syntax Notation One
ASR Automated System Recovery
BAT Bacula Administration Tool
BMR Bare Metal Recovery
BNF Backus–Naur Form
BVFS Bacula Virtual FileSystem
BTRFS B-tree File System
CA Certificate Authority
CBC Cipher Block Chaining
CBT Changed Block Tracking
CD Compact Disk
CDP Continous Data Protection
CDROM Comptact Disk Read Only Memory
CIFS Common Internet File System
CMS Cryptographic Message Syntax
CN Common Name
CRAM Challenge-Response Authentication Mechanism
CSV Comma Separated Value
DC Domain Controller
DDS Digital Data Storage
DER Distinguished Encoding Rules
DFS Distributed File System
DFSR Distributed File System Replication
DH Diffie-Hellman
DHCP Dynamic Host Configuration Protocole
DLT Digital Linear Tape
DNS Domain Name Service
DSA Digital Signature Algorithm
DVD Digital Video Disc
EOF End Of File

FD File Daemon
GNU Gnu is not Unix
GPL Gnu General Public License
GTK Graphic ToolKit
GUI Graphical User Interface
GZIP GNU Zip
HFC High File Count
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
IANA Internet Assigned Numbers Authority
IP Internet Protocol
ISO International Organization for Standardization
KVM Kernel-based Virtual Machine
LAN Local Area Network
LTO Linear Tape Open
LV Logical Volume
LVM Logical Volume Management
LZO Lempel-Ziv-Oberhumer
MCJ Maximum Concurrent Jobs
MD Message Digest
MSDE Microsoft Database Engine
NAS Network Attached Storage
NAT Network Address Translation
NDMP Network Data Management Protocol
NFS Network File System
NIST US National Institute of Standards and Technology
NSIS Nullsoft Scriptable Install System
NTDS NT Directory Service
NTFRS NT File Replication Service
OS Operating System
PEM Privacy Enhanced Mail
PKE Public-Key Encryption
PKI Public-Key Infrastructure
PITR Point In Time Recovery
POSIX Portable Operating System Interface
RAID Redundant Array of Independent Disks
RFC Requests For Comments
RMAN Oracle Recovery Manager
RPM Red Hat Packet Manager
RSA Rivest-Shamir-Adleman cryptosystem
SAN Storage Area Network
SCSI Small Computer System Interface
SD Storage Daemon
SHA Secure Hash Algorithm
SLA Service Level Agreement
SQL Structured Query Language
SSD Solid-State Drive
SSH Secure Shell
SSL Secure Socket Layer

SVN Subversion

TCP Transmission Control Protocol

TLS Transport Layer Security

TLS-PSK Transport Layer Security pre-shared key ciphersuites

UEFI Unified Extensible Firmware Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTF Universal Character Set Transformation Format

VG Volume Group

VSS Volume Snapshot Service

WAN Wide Area Network

WMI Windows Management and Instrumentation

WORM Write Once Read Many

ZFS Z File System

Index

Symbols

A

B

C

D

E

F

K

L

M

N

O

P

Q

R

S

T

U

V

W