It comes in the night and sucks the essence from your computers.

Kern Sibbald

February 6, 2009
This manual documents Bacula® version 2.4.4 (22 January 2009)

Contents

List of Figures

List of Tables

Chapter 1

What is Bacula?

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.

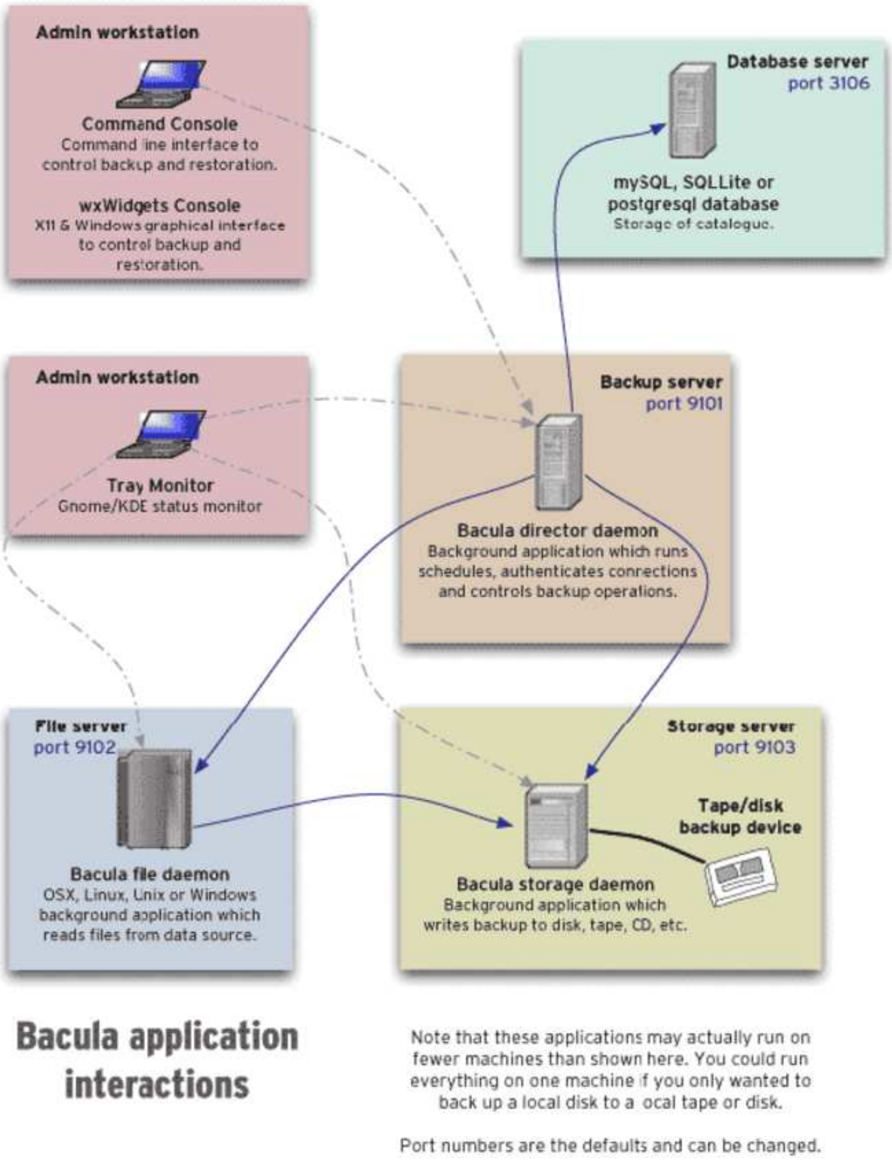
If you want Bacula to behave like the above mentioned simple programs and write over any tape that you put in the drive, then you will find working with Bacula difficult. Bacula is designed to protect your data following the rules you specify, and this means reusing a tape only as the last resort. It 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 are running Amanda and 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 GNU Version 2 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.



(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 Document in the Bacula Developer's Guide. 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, GNOME-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.

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 Document in the Bacula Developer's Guide. 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 Design Document in the Bacula Developer's Guide. 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, and SQLite, one of which must be chosen when building Bacula.

The three SQL databases currently supported (MySQL, PostgreSQL or SQLite) provide 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 and SQLite. For the technical and porting details see the Catalog Services Design Document in the developer's documented.

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.

Configuring and building SQLite is even easier. For the details of configuring SQLite, please see the Installing and Configuring SQLite chapter of this document.

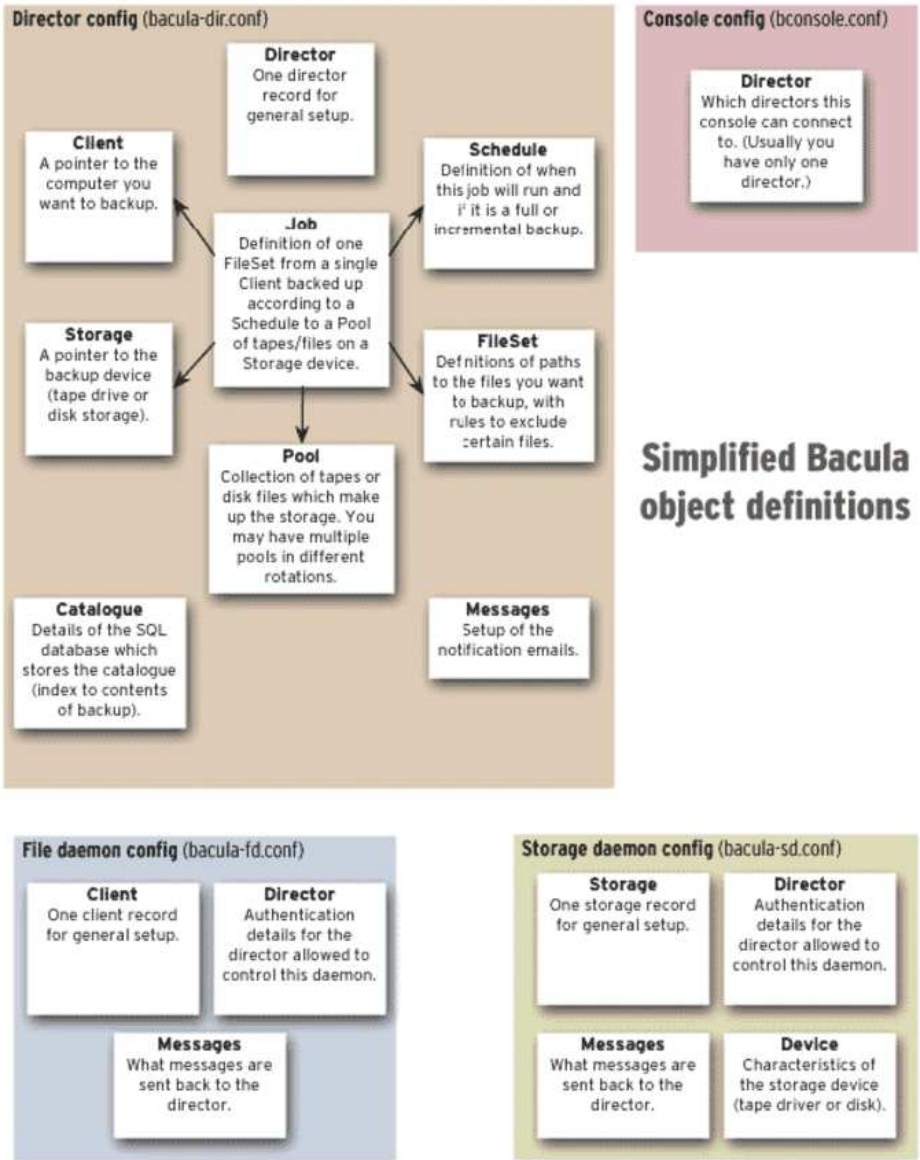
Bacula Monitor

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, PostgreSQL or SQLite).

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:



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 CVS) 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. 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 of the Bacula Utilities Chapter of this manual 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

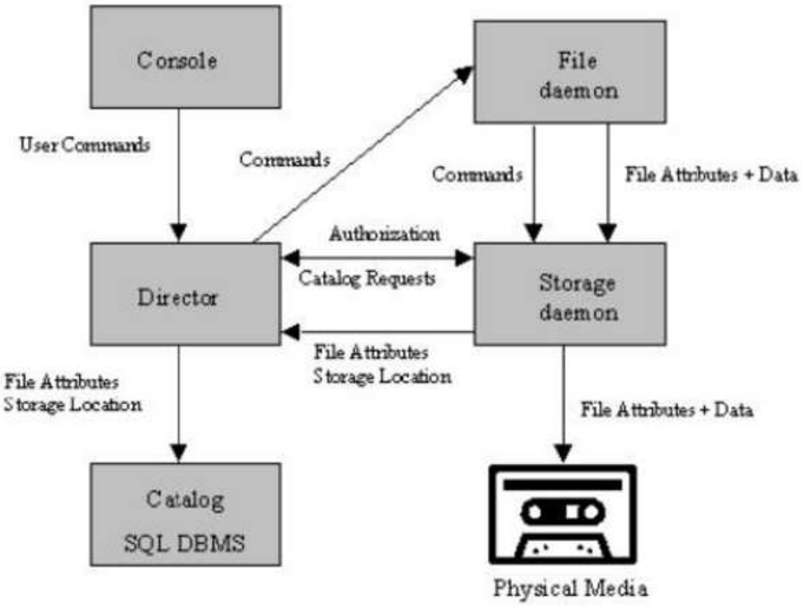
Bacula is a backup, restore and verification program and is not a complete disaster recovery system in itself, but it can be a key part of one if you plan carefully and follow the instructions included in the Disaster Recovery Chapter of this manual.

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, 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 following block diagram 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.


Console
File
daemon
User Commands
Commands
Commands
File Attributes + Data
Authorization
Catalog Requests
Director
Storage
daemon
File Attributes
Storage Location
File Attributes
Storage Location
File Attributes + Data
Catalog
SQL DBMS
Physical Media

Chapter 2

The Current State of Bacula

In other words, what is and what is not currently implemented and functional.

2.1 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, Qt4 GUI, GNOME GUI and wxWidgets GUI versions of the Console program are available. Note, the Qt4 GUI program called the Bacula Administration tool or 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 or 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, PostgreSQL, and SQLite Catalog databases.
  - User extensible queries to the MySQL, PostgreSQL and SQLite 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 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 on version 1.37.28 and greater.
  - 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.

2.2 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.
- 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.

2.3 Current Implementation Restrictions

- If you have over 4 billion file entries stored in your database, the database FileId is likely to overflow. This is a monster database, but still possible. Bacula's FileId fields have been modified so that they can be upgraded from 32 to 64 bits in version 1.39 or later, but you must manually do so.

- Files deleted after a Full save will be included in a restoration. This is typical for most similar backup programs (we have a project to correct this).

- Bacula's Differential and Incremental 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. You must explicitly update the date/time stamp on all moved files (we have a project to correct this).

- File System Modules (configurable routines for saving/restoring special files) are not yet implemented. However, this feature is easily implemented using RunScripts.

- Bacula supports doing backups and restores to multiple devices of different media type and multiple Storage daemons. However, if you have backed up a job to multiple storage devices, Bacula can do a restore from only one device, which means that you will need to manually edit the bootstrap file to split it into two restores if you split the backup across storage devices. This restriction has been removed in version 2.2.0 and later, but it is not yet fully tested.

- Bacula cannot restore two different jobs in the same restore if those jobs were run simultaneously, unless you had data spooling turned on and the spool file held the full contents of both jobs. In other terms, Bacula cannot restore two jobs in the same restore if the jobs' data blocks were intermixed on the backup medium. This poses no restrictions for normal backup jobs even if they are run simultaneously.

- Bacula can generally restore any backup made from a 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).

2.4 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.

Chapter 3

System Requirements

- **Bacula** has been compiled and run on OpenSuSE Linux, FreeBSD, and Solaris 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 4.1
  - PostgreSQL 7.4
  - SQLite 2.8.16 or SQLite 3
- If you want to build the Win32 binaries, please see the README.mingw32 file in the src/win32 directory. We cross-compile the Win32 release on Linux. We provide documentation on building the Win32 version, but due to the complexity, you are pretty much on your own if you want to build it yourself.
- **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 (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.

- 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, apply the patch that is in the **patches** directory of the main source tree to 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 4

Supported Operating Systems

- Linux systems (built and tested on CentOS 5).
- Most flavors of Linux (Gentoo, Red Hat, Fedora, Mandriva, Debian, OpenSuSE, Ubuntu, Kubuntu, ...).
- Solaris various versions.
- FreeBSD (tape driver supported in 1.30 – for FreeBSD older than version 5.0, please see some **important** considerations in the Tape Modes on FreeBSD section of the Tape Testing chapter of this manual.)
- Windows (Win98/Me, WinNT/2K/XP, Vista) Client (File daemon) binaries.
- The Windows servers (Director and Storage daemon) are available in the binary Client installer. The are reported to work in many cases. However they are NOT supported.
- MacOS X/Darwin (see http://fink.sourceforge.net/ for obtaining the packages)
- OpenBSD Client (File daemon).
- Irix Client (File daemon).
- Tru64
- Bacula is said to work on other systems (AIX, BSDI, HPUX, NetBSD, ...) but we do not have first hand knowledge of these systems.

- RHat 7.2 AS2, AS3, AS4, RHEL5, Fedora Core 2,3,4,5,6,7 SuSE SLES 7,8,9,10,10.1,10.2,10.3 and Debian Woody and Sarge Linux on S/390 and Linux on zSeries.
- See the Porting chapter of the Bacula Developer's Guide 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.

Chapter 5

Supported Tape Drives

Yes, Bacula supports your tape drive. Bacula does not directly access hardware. If the tape drive is accessible from your operating system, then it should just work with Bacula. If you can access your tape drive from the command line, then so can Bacula.

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, but we have not tested it. 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-scis, 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.

Even if your drive is on the list below, please check the Tape Testing Chapter of this manual 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:

OS	Man.	Media	Model
-	ADIC	DLT	Adic Scalar 100 DLT
-	ADIC	DLT	Adic Fastor 22 DLT
FreeBSD 5.4-RELEASE-p1 amd64	Certance	LTO	AdicCertance CL400 LTO Ultrium
-	-	DDS	Compaq DDS 2,3,4
SuSE 8.1 Pro	Compaq	AIT	Compaq AIT 35 LVD
-	Exabyte	-	Exabyte drives less than 10 years
-	Exabyte	-	Exabyte VXA drives
-	HP	Travan 4	Colorado T4000S
-	HP	DLT	HP DLT drives
-	HP	LTO	HP LTO Ultrium drives
-	IBM	??	3480, 3480XL, 3490, 3490E, 3580 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
-	Quantum	DLT	DLT-8000
Linux	Seagate	DDS-4	Scorpio 40
FreeBSD 4.9 STABLE	Seagate	DDS-4	STA2401LW
FreeBSD 5.2.1 pthreads patched RELEASE	Seagate	AIT-1	STA1701W
Linux	Sony	DDS-2,3,4	-
Linux	Tandberg	-	Tandbert MLR3
FreeBSD	Tandberg	-	Tandberg SLR6
Solaris	Tandberg	-	Tandberg SLR75

5.1 Don't see your drive?

If you do not see your tape drive listed above, go back and read the first few paragraphs of this page. The list above contains only the hardware that people have reported. If your OS can see the hardware, so can Bacula.

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.

5.2 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.

5.3 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 Chapter of this manual for **important** information on how to configure your tape drive for compatibility with Bacula.

5.4 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.

5.5 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/hr. 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.

Media Type	Drive Type	Media Capacity	Transfer Rate
DDS-1	DAT	2 GB	?? GB/hr
DDS-2	DAT	4 GB	?? GB/hr
DDS-3	DAT	12 GB	5.4 GB/hr
Travan 40	Travan	20 GB	?? GB/hr
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
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 6

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.

6.1 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.

6.2 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.

6.3 Setting Up Bacula Configuration Files

After running the appropriate **./configure** command and doing a **make**, and a **make install**, if this is the first time you are running Bacula, you must create valid configuration files for the Director, the File daemon, the Storage daemon, and the Console programs. If you have followed our recommendations, default configuration files as well as the daemon binaries will be located in your installation directory. In any case, the binaries are found in the directory you specified on the **--sbindir** option to the **./configure** command, and the configuration files are found in the directory you specified on the **--sysconfdir** option.

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 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 changing the tape when it is full.

6.3.1 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**.

If you choose to build the GNOME console with the **--enable-gnome** option, you also find a default configuration file for it, named **bgnome-console.conf**.

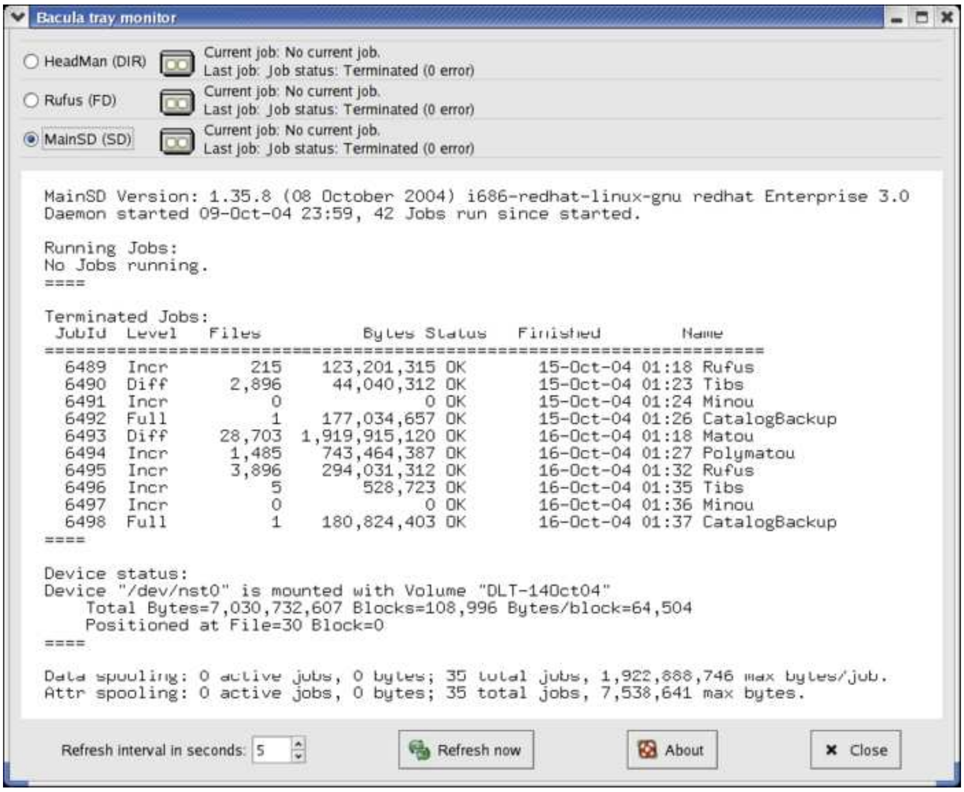
The same applies to the wxWidgets console, which is build with the **--enable-bwx-console** option, and the name of the default configuration file is, in this case, **bwx-console.conf**.

Normally, for first time users, no change is needed to these files. Reasonable defaults are set.

Further details are in the Console configuration chapter.

6.3.2 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.



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.

6.3.3 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.

6.3.4 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.

6.3.5 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.

6.4 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
./bgnome-console -t -c bgnome-console.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).

6.5 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 btape – 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 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.

6.6 Get Rid of the /lib/tls Directory

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.

6.7 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.

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.

6.8 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.

6.9 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.

6.10 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 7

Installing Bacula

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**, **PostgreSQL**, or **SQLite** 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.

7.1 Source Release Files

Beginning with Bacula 1.38.0, the source code has been broken into four separate tar files each corresponding to a different module in the Bacula SVN. The released files are:

bacula-2.0.3.tar.gz This is the primary source code release for Bacula. On each release the version number (2.0.3) will be updated.

bacula-docs-2.0.3.tar.gz This file contains a copy of the docs directory with the documents prebuild. English HTML directory, single HTML file, and pdf file. The French and German translations are in progress, but are not built.

bacula-gui-2.0.3.tar.gz This file contains the non-core GUI programs. Currently, it contains bacula-web, a PHP program for producing management viewing of your Bacula job status in a browser; and bimagemgr a browser program for burning CDROM images with Bacula Volumes.

bacula-rescue-2.0.0.tar.gz This is the Bacula Rescue CDROM code. Note, the version number of this package is not tied to the Bacula release version, so it will be different. Using this code, you can burn a CDROM with your system configuration and containing a statically linked version of the File daemon. This can permit you to easily repartition and reformat your hard disks and reload your system with Bacula in the case of a hard disk failure.

Note, this package evolves slower than the Bacula source code, so there may not always be a new release of the rescue package when making minor updates to the Bacula code. For example, when releasing Bacula version 2.0.3, the rescue package may still be at version 2.0.0 if there were no updates.

winbacula-2.0.3.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. Beginning with Bacula version 1.39.20, this executable will also optionally load the Win32 Director and the Win32 Storage daemon.

7.2 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. 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 there are several database upgrades between your version and the version to which you are upgrading, you will need to apply each database upgrade script. For your convenience, you can find all the old upgrade scripts in the **upgradedb** directory of the source code. You will need to edit the scripts to correspond to your system configuration. The final upgrade script, if any, can be applied as noted above.

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 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 during an upgrade is simply:

```
./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.

For additional information on upgrading, please see the Upgrading Bacula Versions in the Tips chapter of this manual.

7.3 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 (10 September 2006), the current production Bacula release is version 1.38.11. If there are bug fixes, the next release will be 1.38.12 (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 1.38.0 Director with a 1.38.11 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 1.38.11, versions 1.38.0, 1.38.1, ... 1.38.10 have all 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 1.38.11, the current development version is 1.39.22. 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. 1.38.x to 1.40.0), 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 1.38.x or 1.40.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 1.37.x or 1.39.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.

7.4 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-07Apr02.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**
SQLite	X	
SQLite3	X	
mtx	X	
qt4		X
qwt		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 sqlite
```

will configure and build only the SQLite 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.

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 Qt4 using **depkgs-qt**, you must source the **qt4-paths** file included in the package prior to building Bacula. Please read the INSTALL file for more details.

Even if you do not use SQLite, 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.

7.5 Supported Operating Systems

Please see the Supported Operating Systems section of the QuickStart chapter of this manual.

7.6 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.

SQLite is not supported on Solaris. This is because it frequently fails with bus errors. However SQLite3 may work.

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 SQLite Installing and Configuring SQLite Phase II.

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=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --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
```

Note, 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 and editing capability for the Console program. 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 and history 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.

7.7 What Database to Use?

Before building Bacula you need to decide if you want to use SQLite, MySQL, or PostgreSQL. If you are not already running MySQL or PostgreSQL, you might want to start by testing with SQLite (not supported on Solaris). This will greatly simplify the setup for you because SQLite is compiled into Bacula an requires no administration. It performs well and is suitable for small to medium sized installations (maximum 10-20 machines). However, we should note that a number of users have had unexplained database corruption with SQLite. For that reason, we recommend that you install either MySQL or PostgreSQL for production work.

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 any sized installation. It is slightly more complicated than SQLite to setup and administer because it has a number of sophisticated features such as userids and passwords. It runs as a separate process, is truly professional and can manage a database of any size.

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 many more advanced features such as transactions, stored procedures, and the such. It requires a certain knowledge to install and maintain.

If you wish to use SQLite as the Bacula catalog, please see Installing and Configuring SQLite chapter of this manual. SQLite is not supported on Solaris.

7.8 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.

7.9 Configure Options

The following command line options are available for **configure** to customize your installation.

--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.

--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 Qt4 ¿= 4.2 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). In addition to the Qt4 libraries, linking bat requires the qwt package installed on your system. Please see the next configure option (with-qwt) for how to build the qwt package.

Qt4 is available on OpenSUSE 10.2, CentOS 5, 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 and the qwt package, both of which are needed to build bat. See the INSTALL file in that package for more details. In particular to use the Qt4 built by **depkgs-qt** you bf must source the file **qt4-paths**.

--with-qwt=<path> To build bat, you need the qwt graphics package installed on your system. The path specified must be an absolute path and not relative.

The qwt package is available for download from the qwt project on Source Forge. If you wish, you may build and install it on your system (by default in /usr/lib). If you have done so, you would specify:

```
--with-qwt=/usr/lib/qwt-5.0.2
```

Alternatively, you can download the Bacula depkgs package (currently version 11Jul07) and build it, then assuming that you have put it into a directory named bacula, you would specify:

```
--with-qwt=$HOME/bacula/depkgs/qwt
```

Some packages such as Debian do not adhere to the standard of naming the library libqwt.a or libqwt.so, and you will either need to manually add a soft link to the name they use or use the depkgs version, which handles the naming correctly.

--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.

SQLite2 is not thread safe. Batch insert cannot be enabled when using SQLite2

On most systems, MySQL, PostgreSQL and SQLite3 are 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.

As a default, Bacula runs SQLite3 with **PRAGMA synchronous=OFF** because it improves performance by more than 30 times. However, it increases the possibility of a corrupted database. If you want more security, please modify src/version.h appropriately (it should be obvious when you look at the file).

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-gnome If you have GNOME installed on your computer including the GNOME development libraries, and you want to use the GNOME GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/gnome2-console** directory.

--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 GNOME, as wxWidgets can work with GTK+, Motif or even X11 libraries.

--enable-tray-monitor If you have GTK installed on your computer, you run a graphical environment or a window manager compatible with the FreeDesktop system tray standard (like KDE and GNOME) and you want to use a GUI to monitor Bacula daemons, you must specify this option. Doing so will build everything in the **src/tray-monitor** directory. Note, due to restrictions on what can be linked with GPLed code, we were forced to remove the egg code that dealt with the tray icons and replace it by calls to the GTK+ API, and unfortunately, the tray icon API necessary was not implemented until GTK version 2.10 or later.

--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 **--with-python** 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 **--with-python** 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 **--with-python** 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** and a **static-gnome-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 **--with-python** 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 **--with-python** 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-sqlite=<sqlite-path> This enables use of the SQLite version 2.8.x database. The **sqlite-path** is not normally specified as Bacula looks for the necessary components in a standard location (**dep-**

kgs/sqlite). See Installing and Configuring SQLite chapter of this manual for more details. SQLite is not supported on Solaris.

See the note below under the --with-postgresql item.

--with-sqlite3=<sqlite3-path> This enables use of the SQLite version 3.x database. The **sqlite3-path** is not normally specified as Bacula looks for the necessary components in a standard location (**depkgs/sqlite3**). See Installing and Configuring SQLite chapter of this manual for more details. SQLite3 is not supported on Solaris.

--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 four database options supported. That is: --with-sqlite, --with-sqlite3, --with-mysql, or --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. 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.

--with-python=<path> This option enables Bacula support for Python. If no path is supplied, configure will search the standard library locations for Python 2.2, 2.3, 2.4, or 2.5. If it cannot find the library, you will need to supply a path to your Python library directory. Please see the Python chapter for the details of using Python scripting.

--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-base-port=<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.

7.10 Recommended Options for Most Systems

For most systems, we recommend starting with the following options:

```
./configure \
  --enable-smartalloc \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --with-mysql=$HOME/mysql \
  --with-working-dir=$HOME/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.

7.11 Red Hat

Using SQLite:

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --enable-smartalloc \
  --with-sqlite=$HOME/bacula/depkgs/sqlite \
  --with-working-dir=$HOME/bacula/working \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --enable-bat \
  --with-qwt=$HOME/bacula/depkgs/qwt \
  --enable-conio
```

or

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --enable-smartalloc \
  --with-mysql=$HOME/mysql \
  --with-working-dir=$HOME/bacula/working
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working
  --enable-gnome \
  --enable-conio
```

or finally, a completely traditional Red Hat Linux install:

```
CFLAGS="-g -Wall" ./configure \
  --prefix=/usr \
  --sbindir=/usr/sbin \
  --sysconfdir=/etc/bacula \
```

```
  --with-scriptdir=/etc/bacula \
  --enable-smartalloc \
  --enable-bat \
  --with-qwt=$HOME/bacula/depkgs/qwt \
  --with-mysql \
  --with-working-dir=/var/bacula \
  --with-pid-dir=/var/run \
  --enable-conio
```

Note, Bacula assumes that /var/bacula, /var/run, and /var/lock/subsys exist so it will not automatically create them during the install process.

7.12 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=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --with-mysql=$HOME/mysql \
  --enable-smartalloc \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --with-working-dir=$HOME/bacula/working
```

As mentioned above, 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.

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
SUNWbinutilsS
SUNWGmakeS
SUNWlibm

export
PATH=/usr/bin::/usr/ccs/bin:/etc:/usr/openwin/bin:/usr/local/bin:/usr/sfw/bin:/opt/sfw/bin:/usr/
```

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.

7.13 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 this 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.

7.14 Win32

To install the binary Win32 version of the File daemon please see the Win32 Installation Chapter in this document.

7.15 One File Configure Script

The following script could be used if you want to put everything in a single file:

```
#!/bin/sh
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --mandir=$HOME/bacula/bin \
    --enable-smartalloc \
    --enable-gnome \
    --enable-bat \
    --with-qwt=$HOME/bacula/depkgs/qwt \
    --enable-bwx-console \
    --enable-tray-monitor \
    --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
```

7.16 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**.

7.17 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** or **--with-sqlite** options, 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.

7.18 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
```

7.19 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 (e.g. bgnome-console and bgnome-console.conf are not installed if you do not configure GNOME. Also, if you are using SQLite instead of MySQL, some of the files will be different).

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
bacula-tray-monitor
tray-monitor.conf
bextract
bls
bscan
btape
btraceback
btraceback.gdb
bconsole
bconsole.conf
create_mysql_database
dbcheck
delete_catalog_backup
drop_bacula_tables
drop_mysql_tables
bgnome-console
```

```
bgnome-console.conf
make_bacula_tables
make_catalog_backup
make_mysql_tables
mtx-changer
query.sql
bsmtp
startmysql
stopmysql
bwx-console
bwx-console.conf
9 man pages
```

7.20 Installing Tray Monitor

The Tray Monitor is already installed if you used the **--enable-tray-monitor** configure option and ran **make install**.

As you don't run your graphical environment as root (if you do, you should change that bad habit), don't forget to allow your user to read **tray-monitor.conf**, and to execute **bacula-tray-monitor** (this is not a security issue).

Then log into your graphical environment (KDE, GNOME or something else), run **bacula-tray-monitor** as your user, and see if a cassette icon appears somewhere on the screen, usually on the task bar. If it doesn't, follow the instructions below related to your environment or window manager.

7.20.1 GNOME

System tray, or notification area if you use the GNOME terminology, has been supported in GNOME since version 2.2. To activate it, right-click on one of your panels, open the menu **Add to this Panel**, then **Utility** and finally click on **Notification Area**.

7.20.2 KDE

System tray has been supported in KDE since version 3.1. To activate it, right-click on one of your panels, open the menu **Add**, then **Applet** and finally click on **System Tray**.

7.20.3 Other window managers

Read the documentation to know if the Freedesktop system tray standard is supported by your window manager, and if applicable, how to activate it.

7.21 Modifying the Bacula Configuration Files

See the chapter Configuring Bacula in this manual for instructions on how to set Bacula configuration files.

Chapter 8

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.

8.1 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 btape program.

- Better than doing the above is to walk through the nine steps in the Tape Testing chapter of the manual. 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.

- 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 bf 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 modern Win32 machines, you can edit the conf files with **notebook** and choose output encoding UTF-8.

8.2 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 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 floppy 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 9

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.

9.1 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 of this manual. 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.

9.2 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. The scripts **startmysql** and **stopmysql** are what I (Kern) use to start and stop my local MySQL. Note, if you are using SQLite, you will not want to use **startmysql** or **stopmysql**. If you are running this in production, you will probably want to find some way to automatically start MySQL or PostgreSQL after each system reboot.

If you are using SQLite (i.e. you specified the **--with-sqlite=xxx** option on the **./configure** command, you need do nothing. SQLite is automatically started by **Bacula**.

9.3 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.

9.4 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 Bacula Administration Tool (**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 [on|off] -- console messages
  automount  automount [on|off] -- after label
  cancel     cancel [<jobid=nnn> | <job=name>] -- cancel a job
  create     create DB Pool from resource
  delete     delete [pool=<pool-name> | media volume=<volume-name>]
  disable    disable <job=name> -- disable a job
  enable     enable <job=name> -- enable a job
  estimate   performs FileSet estimate, listing gives full listing
  exit       exit = quit
  gui        gui [on|off] -- non-interactive gui mode
  help       print this command
  list       list [pools | jobs | jobtotals | media <pool=pool-name> |
```

```
files <jobid=nn>]; from catalog
  label      label a tape
  llist      full or long list like list command
  memory     print current memory usage
  messages   messages
  mount      mount <storage-name>
  prune      prune expired records from catalog
  purge      purge records from catalog
  python     python control commands
  quit       quit
  query      query catalog
  restore    restore files
  relabel    relabel a tape
  release    release <storage-name>
  reload     reload conf file
  run        run <job-name>
  status     status [storage | client]=<name>
  setdebug   sets debug level
  setip      sets new client address -- if authorized
  show       show (resource records) [jobs | pools | ... | all]
  sqlquery   use SQL to query catalog
  time       print current time
  trace      turn on/off trace to file
  unmount    unmount <storage-name>
  umount     umount <storage-name> for old-time Unix guys
  update     update Volume, Pool or slots
  use        use catalog xxx
  var        does variable expansion
  version    print Director version
  wait       wait until no jobs are running [<jobname=name> | <jobid=nnn> | <ujobid=complete_name>]
*
```

Details of the console program's commands are explained in the Console Chapter of this manual.

9.5 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_sqlite_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:8102
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:8103
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:8103 ...
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.

9.6 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: Cancel
Select item:  (1-12):
```

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** ro **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-restore
```

9.7 Quitting the Console Program

Simply enter the command **quit**.

9.8 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 in the FAQ chapter of this manual.

9.9 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.

9.10 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.

9.11 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.

9.12 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.

9.13 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 number 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).

9.14 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** 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.

-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.

-v Be more verbose or more complete in printing error and informational messages. Recommended.

-? Print the version and list of options.

The Director has the following additional Director specific option:

-r <job> Run the named job immediately. This is for debugging and should not be used.

The File daemon has the following File daemon specific option:

-i Assume that the daemon is called from **inetd** or **xinetd**. In this case, the daemon assumes that a connection has already been made and that it is passed as STDIN. After the connection terminates the daemon will exit.

The Storage daemon has no Storage daemon specific options.

The Console program has no console specific options.

9.15 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 {\bf update} command to
modify many of the values in the Pool record.
```

9.16 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.

9.17 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 of this manual.

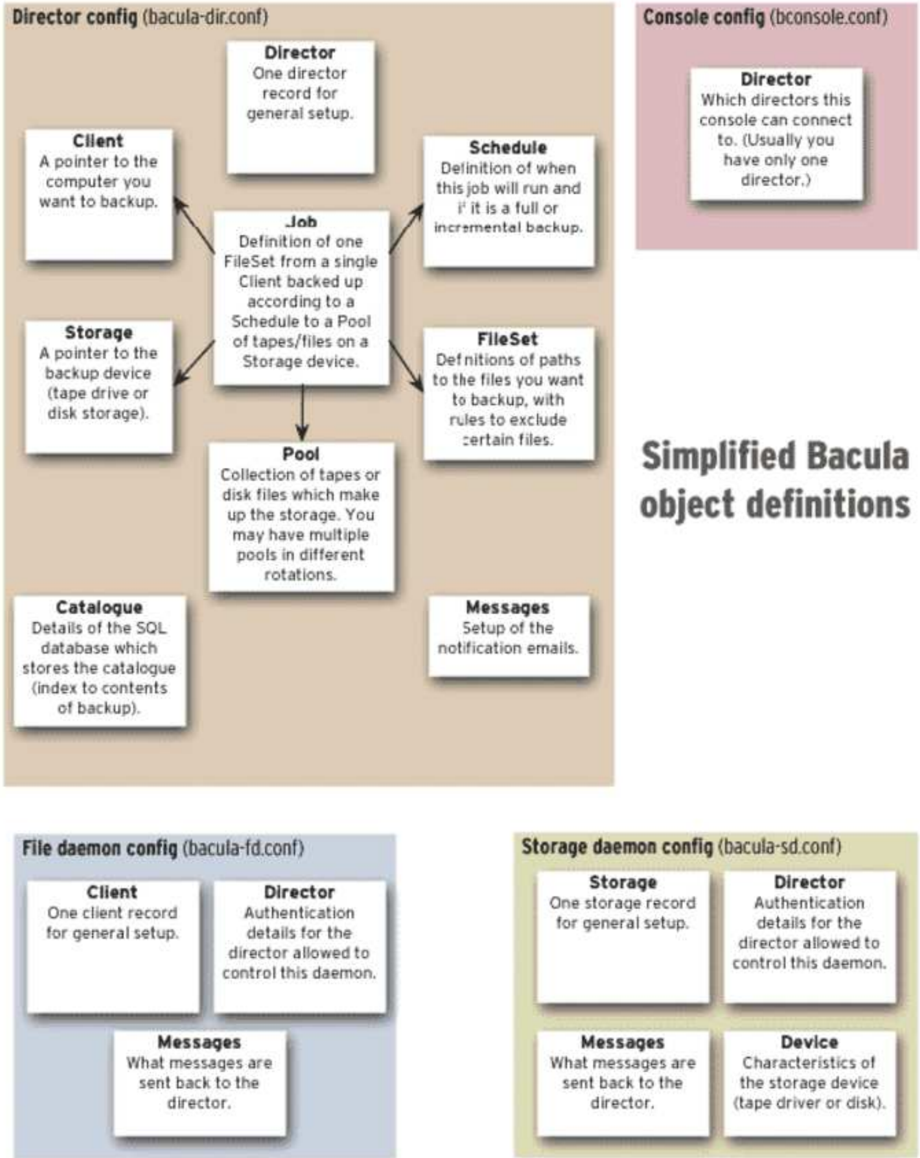
Chapter 10

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:



(thanks to Aristides Maniatis for the above graphic)

10.1 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 bf 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.

10.2 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.

10.2.1 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.

10.2.2 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, ...). All other characters and punctuation are invalid.

10.2.3 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.

10.2.4 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.

10.3 Resource Types

The following table 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.

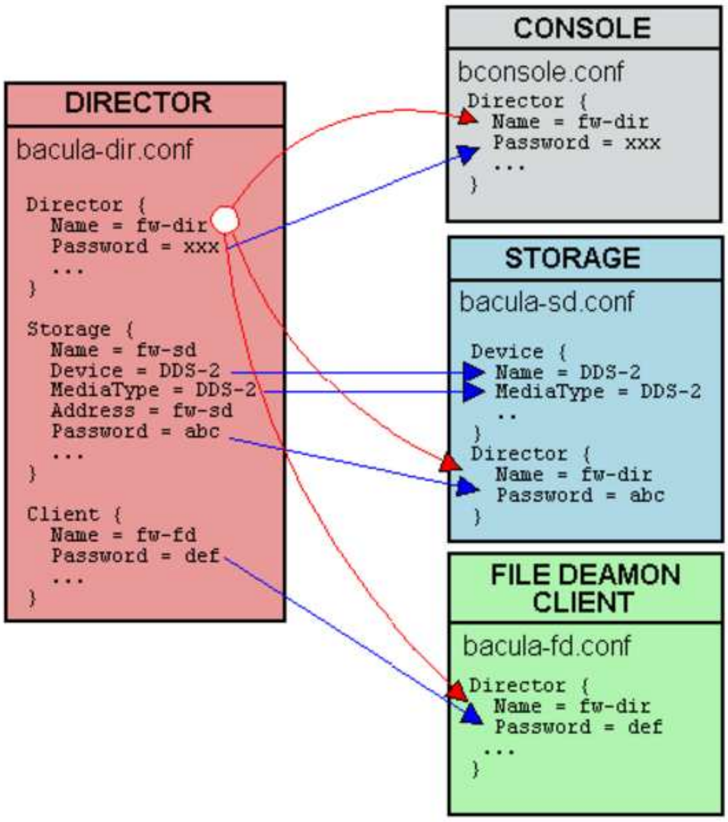
Resource	Director	Client	Storage	Console
Autochanger	No	No	Yes	No
Catalog	Yes	No	No	No
Client	Yes	Yes	No	No
Console	Yes	No	No	Yes
Device	No	No	Yes	No
Director	Yes	Yes	Yes	Yes
FileSet	Yes	No	No	No
Job	Yes	No	No	No
JobDefs	Yes	No	No	No
Message	Yes	Yes	Yes	No
Pool	Yes	No	No	No
Schedule	Yes	No	No	No
Storage	Yes	No	Yes	No

10.4 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:



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.

10.5 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 11

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.

11.1 Director Resource Types

Director resource type may be one of the following:

Job, JobDefs, Client, Storage, Catalog, Schedule, FileSet, Pool, Director, or 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 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 for each Client. 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 – 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. Most people use a single default Pool. However, if you have 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.
- 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, ...).

11.2 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 the final design will 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 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 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 **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 **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 will look for the Python startup script **DirStartup.py**. 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 zero, which means no change is made to the socket.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of total Director Jobs that should run concurrently. The default is set to 1, but you may set it to a larger number.

Please note that the Volume format becomes much more complicated with multiple simultaneous jobs, consequently, restores can take much longer if Bacula must sort through interleaved volume blocks from multiple simultaneous jobs. This can be avoided by having each simultaneously running job write to a different volume or by using data spooling, which will first spool the data to disk simultaneously, then write each spool file to the volume in sequence.

There may also still be some cases where directives such as **Maximum Volume Jobs** are not properly synchronized with multiple simultaneous jobs (subtle timing issues can arise), so careful testing is recommended.

At the current time, there is no configuration parameter set to limit the number of console connections. A maximum of five simultaneous console connections are permitted.

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 30 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 DirAddresses (N.B. 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 (N.B. plural) directive.

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
}
```

11.3 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.

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 automatic execution via the scheduler of a Job.

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. 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. Although an Admin job is recorded in the catalog, very little data is saved.

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).

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. However, to remove deleted files from the catalog during an Incremental backup is quite a time consuming process and not currently implemented in Bacula.

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.

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).

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.

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.

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.

VolumeToCatalog This level causes Bacula to read the file attribute data written to the Volume from the last Job. The file attribute data are compared to the values saved in the Catalog database and any differences are reported. This is similar to the **Catalog** 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 **Catalog** 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.

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 Console 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** file, 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.

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.

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.

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.

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). This directive is implemented in version 1.33 and later.

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 is implemented only in version 1.33 and later.

Incremental Max Wait Time = <time> The time specifies the maximum allowed time that an Incremental backup 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). Please note that if there is a **Max Wait Time** it may also be applied to the job.

Differential Max Wait Time = <time> The time specifies the maximum allowed time that a Differential backup 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). Please note that if there is a **Max Wait Time** it may also be applied to the job.

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). 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.

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 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**.

RunScript {<body-of-runscript>} This directive is implemented in version 1.39.22 and later. 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.

You can use following options may be specified in the body of the runscript:

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 successfu
Runs On Client	Yes/No	*Yes*	Run command on client
Runs When	Before—After—Always	*Never*	When run commands
Fail Job On Error	Yes/No	*Yes*	Fail job if script returns something differe
Command			Path to your script

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:

```
%% = %
%c = Client's name
%d = Director's name
%e = Job Exit Status
%i = JobId
%j = Unique Job id
%l = Job Level
%n = Job name
%s = Since time
%t = Job type (Backup, ...)
%v = Volume name
```

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:

Keyword	RunsOnSuccess	RunsOnFailure	FailJobOnError	Runs On
Run Before Job			Yes	No
Run After Job	Yes	No		No
Run After Failed Job	No	Yes		No
Client Run Before Job			Yes	Yes
Client Run After Job	Yes	No		Yes

Examples:

```
RunScript {
    RunsWhen = Before
    FailJobOnError = No
    Command = "/etc/init.d/apache stop"
}

RunScript {
    RunsWhen = After
    RunsOnFailure = yes
    Command = "/etc/init.d/apache start"
}
```

Special Windows Considerations

In addition, for a Windows client on version 1.33 and above, 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 .conf 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 of this manual.

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 of this manual.

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**.

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. .

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 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 tape shoe-shine (start and stop) during Incremental saves. 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 **no**, which means that the File attributes are sent by the Storage daemon to the Director as they are stored on tape. However, if you want to avoid the possibility that database updates will slow down writing to the tape, you may want to set the value to **yes**, in which case 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.

NOTE: When Spool Data is set to yes, Spool Attributes is also automatically set to yes.

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 backslash (/), 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**:

always when the file to be restored already exists, it is deleted and then replaced by the copy that was backed up.

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 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.

Reschedule Times = <count> This directive specifies the maximum number of times to reschedule the job. If it is set to zero (the default) the job will be rescheduled an indefinite number of times.

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 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.

The default priority is 10.

If you want to run concurrent jobs you should keep these points in mind:

- See Running Concurrent Jobs on how to setup concurrent jobs.
- 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.

Write Part After Job = <yes—no> This directive is only implemented in version 1.37 and later. If this directive is set to **yes** (default **no**), a new part file will be created after the job is finished.

It should be set to **yes** when writing to devices that require mount (for example DVD), so you are sure that the current part, containing this job's data, is written to the device, and that no data is left in the temporary file on the hard disk. However, on some media, like DVD+R and DVD-R, a lot of space (about 10Mb) is lost every time a part is written. So, if you run several jobs each after another, you could set this directive to **no** for all jobs, except the last one, to avoid wasting too much space, but to ensure that the data is written to the medium when all jobs are finished.

This directive is ignored with tape and FIFO devices.

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
}
```

11.4 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.

11.5 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.

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.

SpoolData=yes—no tells Bacula to request the Storage daemon to spool data to a disk file before writing it 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. It prevents interleaving of the job data and reduces or eliminates tape drive stop and start commonly known as "shoe-shine".

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.

WritePartAfterJob=yes—no tells Bacula to request the Storage daemon to write the current part file to the device when the job is finished (see Write Part After Job directive in the Job resource). Please note, this directive is implemented only in version 1.37 and later. The default is yes. We strongly recommend that you keep this set to yes otherwise, when the last job has finished one part will remain in the spool file and restore may or may not work.

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, or fifth 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 | 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
<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-range> | <wday-range> |
                    <daily-keyword>
<day-spec>        = <day> | <wday-keyword> |
                    <day> | <wday-range> |
                    <week-keyword> <wday-keyword> |
                    <week-keyword> <wday-range>
<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 NIST (US National Institute of Standards and Technology), 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
}
```

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
}
```

11.6 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).

-

11.7 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 **notebook** and choose output encoding UTF-8.

To ensure that Bacula configuration files can be correctly read including foreign characters the bf 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.

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.

The default is **no**, in which case, if you change the Include or Exclude, 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 Shadow Copy 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 away, Bacula can at least copy open files. For more information, please see the Windows chapter of this manual.

Include { Options {<file-options>} ...; <file-list> }

Options { <file-options> }

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 in lower case (e.g. **c:/xxx**) using Unix directory name separators (forward slash).

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 {
  File = /
  File = /usr
```

```
    Options { compression=GZIP }
  }
```

on a Unix system where /usr is a subdirectory (rather than a mounted filesystem) will cause /usr to be backed up twice. In this case, on Bacula versions prior to 1.32f-5-09Mar04 due to a bug, you will not be able to restore hard linked files that were backed up twice.

If you have used Bacula prior to version 1.36.3, you will note three things in the new 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.

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.

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=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.

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

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

A useful set of general options on the **Level=Catalog** or **Level=DiskToCatalog** verify is **pins5** i.e. compare permission bits, inodes, number of links, size, and MD5 changes.

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
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 previous 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.

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 API calls and on WinNT/2K/XP, 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 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 a small additional amount of space on the output archive will be used to save the seek address of each non-zero record read.

Restrictions: Bacula reads files in 32K 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 32K 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.

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 (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 On versions 2.0.4 or greater, if enabled, the Client will checks 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.

```
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.

hardlinks=yes—no When enabled (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 of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual. It is recommended to enclose the string in double quotes.

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 of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this 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 of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this 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 of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual.

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 Utilities chapter of this manual for 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 Utilities chapter of this manual for 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 system, Bacula will backup the file and directory UNIX Access Control Lists (ACL) as defined in IEEE Std 1003.1e draft 17 and ”POSIX.1e” (abandoned). This feature is available on UNIX only and depends on the ACL library. Bacula is automatically compiled with ACL support if the **libacl** library is installed on your 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.

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 This option allows you to select files and directories by the filesystem type. The permitted filesystem-type names are:

ext2, jfs, ntfs, proc, reiserfs, xfs, usbdevfs, sysfs, smbfs, iso9660. For ext3 systems, use ext2.

You may have multiple Fstype directives, and thus permit matching of multiple filesystem types within a single Options resource. If the type specified on the fstype 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 Win32 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 = /
}
```

- 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.

  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.

11.8 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
  }
}
```

You can add wild-cards to the File directives listed in the Exclude directory, 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 lets take a slight variation on the above and suppose you want to save all your whole filesystem 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 entries 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
  }
}
```

11.9 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.

11.10 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. For example:

```
FileSet {
  Name = Exclusion_example
  Include {
    Options {
      Signature = SHA1
    }
    File = /
    File = /boot
    File = /home
    File = /rescue
    File = /usr
  }
```

```
  Exclude {
    File = /proc
    File = /tmp
    File = .journal
    File = .autofsck
  }
}
```

11.11 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 2 exclude a directory, you must not have a trailing slash on the directory name.
- I2 f 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 Console chapter of this 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.

11.12 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 in the Console chapter of this 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.

11.13 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.

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 is a port number at which the Bacula File server daemon can be contacted. The default is 9102.

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 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.

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. We strongly recommend that you read the WARNING documented under Maximum Concurrent Jobs in the Director's resource.

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).

The following is an example of a valid Client resource definition:

```
Client {
  Name = Minimatou
  FDAddress = minimatou
  Catalog = MySQL
  Password = very_good
}
```

11.14 The Storage Resource

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.

Address = <address> Where the address is a host name, a **fully qualified domain name**, 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.

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 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.

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.

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 zero, which means no change is made to the socket.

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
}
```

11.15 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.

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 (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 but it can be recycled if recycling is enabled, and thus used again. By setting **MaximumVolumeJobs** to one, 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 (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 (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 **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 the Volume can be re-used after recycling. This value is checked and the **Used** status set while the job is writing to the particular volume.

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 (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 will not work correctly (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 in the Console.

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 length 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. 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 retentions 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. This pruning could 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, 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.

RecyclePool = <pool-resource-name> On versions 2.1.4 or greater, 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 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. 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 Append, Full, Used, or Error, 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**.

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 of this manual.

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 the Console Chapter of this manual.

In almost all cases, you should enclose the format specification (part after the equal sign) in double quotes. Please note that this directive is deprecated and is replaced in version 1.37 and greater with a Python script for creating volume names.

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 Console**, 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
}
```

11.15.1 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.

11.16 The Catalog Resource

The Catalog Resource defines what catalog to use for the current job. Currently, Bacula can only handle a single database server (SQLite, 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 SQLite is compiled in, it always runs on the same machine as the Director and the database must be directly accessible (mounted) from the Director. However, 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 and is ignored by SQLite. 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 and is ignored by SQLite if provided. 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 and is ignored by SQLite if provided. This directive is optional.

the different

The following is an example of a valid Catalog resource definition:

```
Catalog
{
  Name = SQLite
```

```
  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
}
```

11.17 The Messages Resource

For the details of the Messages Resource, please see the Messages Resource Chapter of this manual.

11.18 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 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.

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.

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 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.

11.19 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.

11.20 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 12

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.

12.1 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 Director 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.

Heartbeat Interval = <time-interval> This record defines an interval of time. 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 zero which disables the heartbeat. 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 2, 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.

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).

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.

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.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the File daemon to set a keepalive interval (heartbeat) in seconds on each of the sockets to communicate with the Storage daemon. 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.

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
}
```

12.2 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.

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.

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
}
```

12.3 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.

12.4 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 = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZ0bxT3XQxgxppTn"
```

```
  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 13

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.

13.1 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 Director 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.

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 zero which disables the heartbeat. 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 seconds. 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 should run concurrently. The default is set to 10, 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).

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"
}
```

13.2 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.

The following is an example of a valid Director resource definition:

```
Director {
  Name = MainDirector
  Password = my_secret_password
}
```

13.3 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**". For a DVD-writer, it will be for example **/dev/hdc**. 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.

DVD The device is a DVD. DVDs are sequential access for writing, but random access for reading.

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, which means that on those systems, Bacula is unlikely to be able to correctly guess that a device is a DVD. 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.

This directive was implemented in Bacula version 1.38.6.

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**.

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.

Note, it is not necessary to have an autochanger to use this command. The example below uses the **tapeinfo** program that comes with the **mtx** package, but it can be used on any tape drive. However, you will need to specify a **Changer Device** directive in your Device resource (see above) so that the generic SCSI device name can be edited into the command (with the %c).

An example of the use of this command to print Tape Alerts in the Job report is:

```
Alert Command = "sh -c 'tapeinfo -f %c | grep TapeAlert'"
```

and an example output when there is a problem could be:

```
bacula-sd  Alert: TapeAlert[32]: Interface: Problem with SCSI interface
                  between tape drive and initiator.
```

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 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 for Bacula to wait for a open before timing out. If this time is exceeded, Bacula will cancel the job. The default is 5 minutes.

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 (the default), no polling will occur. 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 in the Tape Testing chapter.

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.

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 DVD, 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 DVD-writers and 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 a DVD, the following directives must also be defined: **Mount Point**, **Mount Command**, **Unmount Command** and **Write Part 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 DVD or USB file devices.

Mount Command = *name-string* This directive specifies the command that must be executed to mount devices such as DVDs and many USB devices. For DVDs, the device is written directly, but the mount command is necessary in order to determine the free space left on the DVD. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, for a DVD, you will define it as follows:

```
Mount Command = "/bin/mount -t iso9660 -o ro %a %m"
```

However, if you have defined a mount point in /etc/fstab, you might be able to use a mount command such as:

```
Mount Command = "/bin/mount /media/dvd"
```

See the Edit Codes section below for more details of the editing codes that can be used in this directive.

Unmount Command = *name-string* This directive specifies the command that must be executed to unmount devices such as DVDs and 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.

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 = 200K
```

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 use fixed block sizes (see Minimum block size above). 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 exceed 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).

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 in the Tape Testing chapter.

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, DVD, ...) 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.

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.

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 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 (DVD). The default is to use the working directory.

Maximum Part Size = *bytes* This is the maximum size of a volume part file. The default is no limit. This directive is implemented only in version 1.37 and later.

If the device requires mount, it is transferred to the device when this size is reached. In this case, you must take care to have enough disk space left in the spool directory.

Otherwise, it is left on the hard disk.

It is ignored for tape and FIFO devices.

13.4 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)
```

13.5 Devices that require a mount (DVD)

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 in addition to DVD.

Requires Mount = *Yes—No* You must set this directive to **yes** for DVD-writers, 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** and **Write Part 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"
```

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"
```

Write Part Command = *name-string* Command that must be executed to write a part to the device. Before the command is executed, %a is replaced with the Archive Device, %m with the Mount Point, %e is replaced with 1 if we are writing the first part, and with 0 otherwise, and %v with the current part filename.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Write Part Command = "/path/dvd-handler %a write %e %v"
```

Where **/path** is the path to your scripts install directory, and dvd-handler is the Bacula supplied script file. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

Free Space Command = *name-string* Command that must be executed to check how much free space is left on the device. Before the command is executed,%a is replaced with the Archive Device, %m with the Mount Point, %e is replaced with 1 if we are writing the first part, and with 0 otherwise, and %v with the current part filename.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Free Space Command = "/path/dvd-handler %a free"
```

Where **/path** is the path to your scripts install directory, and dvd-handler is the Bacula supplied script file. If you want to specify your own command, please look at the code of dvd-handler to see what output Bacula expects from this command. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

If you do not set it, Bacula will expect there is always free space on the device.

Chapter 14

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 = "/etc/bacula/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.

14.1 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.

14.2 Messages Resource

For a description of the Messages Resource, please see the Messages Resource Chapter of this manual.

14.3 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 = "ZF9Ctf5PQoWCPkmR3s4atCB0usUPg+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 = "/home/kern/bacula/bin/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;
#}
#Device {
```

```
#  Name = "NEC ND-1300A"
#  Media Type = DVD
#  Archive Device = /dev/hda
#  LabelMedia = yes;                   # lets Bacula label unlabeled media
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  RemovableMedia = yes;
#  AlwaysOpen = no;
#  MaximumPartSize = 800M;
#  RequiresMount = yes;
#  MountPoint = /mnt/cdrom;
#  MountCommand = "/bin/mount -t iso9660 -o ro %a %m";
#  UnmountCommand = "/bin/umount %m";
#  SpoolDirectory = /tmp/backup;
#  WritePartCommand = "/etc/bacula/dvd-handler %a write %e %v"
#  FreeSpaceCommand = "/etc/bacula/dvd-handler %a free"
#}
#
# 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 15

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. Using the **MailCommand**, you can specify exactly how to send the mail. During the processing of the **command**, normally specified as a quoted string, the following substitutions will be used:

- %% = %
- %c = Client's name
- %d = Director's name
- %e = Job Exit code (OK, Error, ...)
- %i = Job Id
- %j = Unique Job name
- %l = Job level
- %n = Job name
- %r = Recipients

- %t = Job type (e.g. Backup, ...)

The following is the command I (Kern) use. Note, the whole command should appear on a single line in the configuration file rather than split as is done here for presentation:

mailcommand = "/home/kern/bacula/bin/bsmtp -h mail.example.com -f \"\(Bacula\) %r\" -s \"Bacula: %t %e of %c %l\" %r"

Note, the **bsmtp** program is provided as part of **Bacula**. For additional details, please see the bsmtp – Customizing Your Email Messages section of the Bacula Utility Programs chapter of this manual. 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.

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**.

<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
```

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 in 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.

stdout Send the message to the standard output (normally not used).

stderr Send the message to the standard error output (normally not used).

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**).

For any destination, the **message-type** field is a comma separated list of the following types or classes of messages:

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.

all All message types.

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
}
```

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/bin/log" = all, !skipped, !terminate
  operator = security@example.com = mount
  console = all, !skipped, !saved
}
```

Chapter 16

Console Configuration

16.1 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:

16.2 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-base-port** 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.

An actual example might be:

```
Director {
  Name = HeadMan
  address = rufus.cats.com
  password = xyz1erploit
}
```

16.3 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"
}
```

16.4 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 resource. 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" 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.

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 = "XXXXXXXXXXX"    # 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 = "XXXXXXXXXXX"    # no, really.  this is not obfuscation.
}

Director {
   Name = SecondDirector
   DIRport = 9101
   Address = secondserver
   Password = "XXXXXXXXXXX"    # no, really.  this is not obfuscation.
```

```
}

Console {
   Name = restricted-user
   Password = "UntrustedUser"
   Director = MyDirector
}

Console {
   Name = restricted-user
   Password = "A different UntrustedUser"
   Director = SecondDirector
}
```

The second Director referenced at ”secondserver” might look like the following:

```
Console {
  Name = restricted-user
  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 = "/"
}
```

16.5 Console Commands

For more details on running the console and its commands, please see the Bacula Console chapter of this manual.

16.6 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 17

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.

17.1 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.

Password = <password> Where the password is the password needed for Directors to accept the Console connection. This password must be identical to the **Password** specified in the **Console** resource of the Director's configuration file. This record is required if you wish to monitor Directors.

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.

17.2 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.

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-base-port** 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.

17.3 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.

FD 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.

17.4 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.

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 record is required.

17.5 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).

17.6 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
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  RefreshInterval = 10 seconds
}

Client {
  Name = rufus-fd
  Address = rufus
  FDPort = 9102           # password for FileDaemon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
}
Storage {
  Name = rufus-sd
  Address = rufus
  SDPort = 9103           # password for StorageDaemon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
}
Director {
  Name = rufus-dir
  DIRport = 9101
  address = rufus
}
```

17.6.1 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 = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
  Monitor = yes
}
```

17.6.2 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
}
```

17.6.3 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 18

Bacula Console

The **Bacula Console** (sometimes called the User Agent) is a program that allows the user or the System Administrator, to interact with the Bacula Director daemon while the daemon is running.

The current Bacula Console comes in two versions: a shell interface (TTY style), and a GNOME GUI interface. Both permit the administrator or authorized users to interact with Bacula. You can determine the status of a particular job, examine the contents of the Catalog as well as perform certain tape manipulations with the Console program.

In addition, there is a bwx-console built with wxWidgets that allows a graphic restore of files. As of version 1.34.1 it is in an early stage of development, but it already is quite useful. Unfortunately, it has not been enhanced for some time now.

Since the Console program interacts with the Director through the network, your Console and Director programs do not necessarily need to run on the same machine.

In fact, a certain minimal knowledge of the Console program is needed in order for Bacula to be able to write on more than one tape, because when Bacula requests a new tape, it waits until the user, via the Console program, indicates that the new tape is mounted.

18.1 Console Configuration

When the Console starts, it reads a standard Bacula configuration file named **bconsole.conf** or **bgnome-console.conf** in the case of the GNOME Console version. This file allows default configuration of the Console, and at the current time, the only Resource Record defined is the Director resource, which gives the Console the name and address of the Director. For more information on configuration of the Console program, please see the Console Configuration File Chapter of this document.

18.2 Running the Console Program

The console program can be run with the following options:

```
Usage: bconsole [-s] [-c config_file] [-d debug_level]
       -c <file>   set configuration file to file
       -dnn        set debug level to nn
       -n          no conio
       -s          no signals
       -t          test - read configuration and exit
       -?          print this message.
```

After launching the Console program (bconsole), it will prompt you for the next command with an asterisk (*). (Note, in the GNOME version, the prompt is not present; you simply enter the commands you want in the command text box at the bottom of the screen.) Generally, for all commands, you can simply enter the command name and the Console program will prompt you for the necessary arguments. Alternatively, in most cases, you may enter the command followed by arguments. The general format is:

```
 <command> <keyword1>[=<argument1>] <keyword2>[=<argument2>] ...
```

where **command** is one of the commands listed below; **keyword** is one of the keywords listed below (usually followed by an argument); and **argument** is the value. The command may be abbreviated to the shortest unique form. If two commands have the same starting letters, the one that will be selected is the one that appears first in the **help** listing. If you want the second command, simply spell out the full command. None of the keywords following the command may be abbreviated.

For example:

```
list files jobid=23
```

will list all files saved for JobId 23. Or:

```
show pools
```

will display all the Pool resource records.

The maximum command line length is limited to 511 characters, so if you are scripting the console, you may need to take some care to limit the line length.

18.3 Stopping the Console Program

Normally, you simply enter **quit** or **exit** and the Console program will terminate. However, it waits until the Director acknowledges the command. If the Director is already doing a lengthy command (e.g. prune), it may take some time. If you want to immediately terminate the Console program, enter the **.quit** command.

There is currently no way to interrupt a Console command once issued (i.e. Ctrl-C does not work). However, if you are at a prompt that is asking you to select one of several possibilities and you would like to abort the command, you can enter a period (**.**), and in most cases, you will either be returned to the main command prompt or if appropriate the previous prompt (in the case of nested prompts). In a few places such as where it is asking for a Volume name, the period will be taken to be the Volume name. In that case, you will most likely be able to cancel at the next prompt.

18.4 Alphabetic List of Console Keywords

Unless otherwise specified, each of the following keywords takes an argument, which is specified after the keyword following an equal sign. For example:

```
jobid=536
```

Please note, this list is incomplete as it is currently in the process of being created and is not currently totally in alphabetic order ...

restart Permitted on the python command, and causes the Python interpreter to be restarted. Takes no argument.

all Permitted on the status and show commands to specify all components or resources respectively.

allfrompool Permitted on the update command to specify that all Volumes in the pool (specified on the command line) should be updated.

allfrompools Permitted on the update command to specify that all Volumes in all pools should be updated.

before Used in the restore command.

bootstrap Used in the restore command.

catalog Allowed in the use command to specify the catalog name to be used.

catalogs Used in the show command. Takes no arguments.

client — fd

clients Used in the show, list, and llist commands. Takes no arguments.

counters Used in the show command. Takes no arguments.

current Used in the restore command. Takes no argument.

days Used to define the number of days the "list nextvol" command should consider when looking for jobs to be run. The days keyword can also be used on the "status dir" command so that it will display jobs scheduled for the number of days you want.

devices Used in the show command. Takes no arguments.

dir — director

directors Used in the show command. Takes no arguments.

directory Used in the restore command. Its argument specifies the directory to be restored.

enabled This keyword can appear on the **update volume** as well as the **update slots** commands, and can allows one of the following arguments: yes, true, no, false, archived, 0, 1, 2. Where 0 corresponds to no or false, 1 corresponds to yes or true, and 2 corresponds to archived. Archived volumes will not be used, nor will the Media record in the catalog be pruned. Volumes that are not enabled, will not be used for backup or restore.

done Used in the restore command. Takes no argument.

file Used in the restore command.

files Used in the list and llist commands. Takes no arguments.

fileset

filesets Used in the show command. Takes no arguments.

help Used in the show command. Takes no arguments.

jobs Used in the show, list and llist commands. Takes no arguments.

jobmedia Used in the list and llist commands. Takes no arguments.

jobtotals Used in the list and llist commands. Takes no arguments.

jobid The JobId is the numeric jobid that is printed in the Job Report output. It is the index of the database record for the given job. While it is unique for all the existing Job records in the catalog database, the same JobId can be reused once a Job is removed from the catalog. Probably you will refer specific Jobs that ran using their numeric JobId.

job — jobname The Job or Jobname keyword refers to the name you specified in the Job resource, and hence it refers to any number of Jobs that ran. It is typically useful if you want to list all jobs of a particular name.

level

listing Permitted on the estimate command. Takes no argument.

limit

messages Used in the show command. Takes no arguments.

media Used in the list and llist commands. Takes no arguments.

nextvol — nextvolume Used in the list and llist commands. Takes no arguments.

on Takes no keyword.

off Takes no keyword.

pool

pools Used in the show, list, and llist commands. Takes no arguments.

select Used in the restore command. Takes no argument.

storages Used in the show command. Takes no arguments.

schedules Used in the show command. Takes no arguments.

sd — store — storage

ujobid The ujobid is a unique job identification that is printed in the Job Report output. At the current time, it consists of the Job name (from the Name directive for the job) appended with the date and time the job was run. This keyword is useful if you want to completely identify the Job instance run.

volume

volumes Used in the list and llist commands. Takes no arguments.

where Used in the restore command.

yes Used in the restore command. Takes no argument.

18.5 Alphabetic List of Console Commands

The following commands are currently implemented:

add [pool=<pool-name> storage=<storage> jobid=<JobId>] This command is used to add Volumes to an existing Pool. That is, it creates the Volume name in the catalog and inserts into the Pool in the catalog, but does not attempt to access the physical Volume. Once added, Bacula expects that Volume to exist and to be labeled. This command is not normally used since Bacula will automatically do the equivalent when Volumes are labeled. However, there may be times when you have removed a Volume from the catalog and want to later add it back.

Normally, the **label** command is used rather than this command because the **label** command labels the physical media (tape, disk, DVD, ...) and does the equivalent of the **add** command. The **add** command affects only the Catalog and not the physical media (data on Volumes). The physical media must exist and be labeled before use (usually with the **label** command). This command can, however, be useful if you wish to add a number of Volumes to the Pool that will be physically labeled at a later time. It can also be useful if you are importing a tape from another site. Please see the **label** command below for the list of legal characters in a Volume name.

autodisplay on/off This command accepts **on** or **off** as an argument, and turns auto-display of messages on or off respectively. The default for the console program is **off**, which means that you will be notified when there are console messages pending, but they will not automatically be displayed. The default for the bgnome-console program is **on**, which means that messages will be displayed when they are received (usually within five seconds of them being generated).

When autodisplay is turned off, you must explicitly retrieve the messages with the **messages** command. When autodisplay is turned on, the messages will be displayed on the console as they are received.

automount on/off This command accepts **on** or **off** as the argument, and turns auto-mounting of the Volume after a **label** command on or off respectively. The default is **on**. If **automount** is turned off, you must explicitly **mount** tape Volumes after a label command to use it.

cancel [jobid=<number> job=<job-name> ujobid=<unique-jobid>] This command is used to cancel a job and accepts **jobid=nnn** or **job=xxx** as an argument where nnn is replaced by the JobId and xxx is replaced by the job name. If you do not specify a keyword, the Console program will prompt you with the names of all the active jobs allowing you to choose one.

Once a Job is marked to be canceled, it may take a bit of time (generally within a minute) before it actually terminates, depending on what operations it is doing.

create [pool=<pool-name>] This command is not normally used as the Pool records are automatically created by the Director when it starts based on what it finds in the conf file. If needed, this command can be to create a Pool record in the database using the Pool resource record defined in the Director's configuration file. So in a sense, this command simply transfers the information from the Pool resource in the configuration file into the Catalog. Normally this command is done automatically for you when the Director starts providing the Pool is referenced within a Job resource. If you use this command on an existing Pool, it will automatically update the Catalog to have the same information as the Pool resource. After creating a Pool, you will most likely use the **label** command to label one or more volumes and add their names to the Media database.

When starting a Job, if Bacula determines that there is no Pool record in the database, but there is a Pool resource of the appropriate name, it will create it for you. If you want the Pool record to appear in the database immediately, simply use this command to force it to be created.

delete [**volume=<vol-name> pool=<pool-name> job jobid=<id>**] The delete command is used to delete a Volume, Pool or Job record from the Catalog as well as all associated catalog Volume records that were created. This command operates only on the Catalog database and has no effect on the actual data written to a Volume. This command can be dangerous and we strongly recommend that you do not use it unless you know what you are doing.

If the keyword **Volume** appears on the command line, the named Volume will be deleted from the catalog, if the keyword **Pool** appears on the command line, a Pool will be deleted, and if the keyword **Job** appears on the command line, a Job and all its associated records (File and JobMedia) will be deleted from the catalog. The full form of this command is:

```
delete pool=<pool-name>
```

or

```
delete volume=>volume-name> pool=>pool-name>  or

delete JobId=>job-id> JobId=>job-id2> ...  or

delete Job JobId=n,m,o-r,t ...
```

The first form deletes a Pool record from the catalog database. The second form deletes a Volume record from the specified pool in the catalog database. The third form deletes the specified Job record from the catalog database. The last form deletes JobId records for JobIds n, m, o, p, q, r, and t. Where each one of the n,m,... is, of course, a number. That is a "delete jobid" accepts lists and ranges of jobids.

disable job<job-name> This command permits you to disable a Job for automatic scheduling. The job may have been previously enabled with the Job resource **Enabled** directive or using the console **enable** command. The next time the Director is restarted or the conf file is reloaded, the Enable/Disable state will be set to the value in the Job resource (default enabled) as defined in the bacula-dir.conf file.

enable job<job-name> This command permits you to enable a Job for automatic scheduling. The job may have been previously disabled with the Job resource **Enabled** directive or using the console **disable** command. The next time the Director is restarted or the conf file is reloaded, the Enable/Disable state will be set to the value in the Job resource (default enabled) as defined in the bacula-dir.conf file.

estimate Using this command, you can get an idea how many files will be backed up, or if you are unsure about your Include statements in your FileSet, you can test them without doing an actual backup. The default is to assume a Full backup. However, you can override this by specifying a **level=Incremental** or **level=Differential** on the command line. A Job name must be specified or you will be prompted for one, and optionally a Client and FileSet may be specified on the command line. It then contacts the client which computes the number of files and bytes that would be backed up. Please note that this is an estimate calculated from the number of blocks in the file rather than by reading the actual bytes. As such, the estimated backup size will generally be larger than an actual backup.

Optionally you may specify the keyword **listing** in which case, all the files to be backed up will be listed. Note, it could take quite some time to display them if the backup is large. The full form is:

```
estimate job=<job-name> listing client=<client-name>
       fileset=<fileset-name> level=<level-name>
```

Specification of the **job** is sufficient, but you can also override the client, fileset and/or level by specifying them on the estimate command line.

As an example, you might do:

```
@output /tmp/listing
estimate job=NightlySave listing level=Incremental
@output
```

which will do a full listing of all files to be backed up for the Job **NightlySave** during an Incremental save and put it in the file **/tmp/listing**. Note, the byte estimate provided by this command is based on the file size contained in the directory item. This can give wildly incorrect estimates of the actual storage used if there are sparse files on your systems. Sparse files are often found on 64 bit systems for certain system files. The size that is returned is the size Bacula will backup if the sparse option is not specified in the FileSet. There is currently no way to get an estimate of the real file size that would be found should the sparse option be enabled.

help This command displays the list of commands available.

label This command is used to label physical volumes. The full form of this command is:

```
label storage=>storage-name> volume=>volume-name>
      slot=>slot>
```

If you leave out any part, you will be prompted for it. The media type is automatically taken from the Storage resource definition that you supply. Once the necessary information is obtained, the Console program contacts the specified Storage daemon and requests that the Volume be labeled. If the Volume labeling is successful, the Console program will create a Volume record in the appropriate Pool.

The Volume name is restricted to letters, numbers, and the special characters hyphen (**-**), underscore (**_**), colon (**:**), and period (**.**). All other characters including a space are invalid. This restriction is to ensure good readability of Volume names to reduce operator errors.

Please note, when labeling a blank tape, Bacula will get **read I/O error** when it attempts to ensure that the tape is not already labeled. If you wish to avoid getting these messages, please write an EOF mark on your tape before attempting to label it:

```
mt rewind
mt weof
```

The label command can fail for a number of reasons:

1. The Volume name you specify is already in the Volume database.
2. The Storage daemon has a tape or other Volume already mounted on the device, in which case you must **unmount** the device, insert a blank tape, then do the **label** command.
3. The Volume in the device is already a Bacula labeled Volume. (Bacula will never relabel a Bacula labeled Volume unless it is recycled and you use the **relabel** command).
4. There is no Volume in the drive.

There are two ways to relabel a volume that already has a Bacula label. The brute force method is to write an end of file mark on the tape using the system **mt** program, something like the following:

```
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

For a disk volume, you would manually delete the Volume.

Then you use the **label** command to add a new label. However, this could leave traces of the old volume in the catalog.

The preferable method to relabel a Volume is to first **purge** the volume, either automatically, or explicitly with the **purge** command, then use the **relabel** command described below.

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" directive in the Director's Pool resource, will be treated as a cleaning tape, and will not be labeled. However, an entry for the cleaning tape will be created in the catalog. 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. Note, the full form of the command is:

```
label storage=xxx pool=yyy slots=1-5,10 barcodes
```

list The list command lists the requested contents of the Catalog. The various fields of each record are listed on a single line. The various forms of the list command are:

```
list jobs

list jobid=<id>           (list jobid id)

list ujobid<unique job name> (list job with unique name)

list job=<job-name>   (list all jobs with "job-name")

list jobname=<job-name>  (same as above)

    In the above, you can add "limit=nn" to limit the output to
    nn jobs.

list jobmedia
```

```
list jobmedia jobid=<id>

list jobmedia job=<job-name>

list files jobid=<id>

list files job=<job-name>

list pools

list clients

list jobtotals

list volumes

list volumes jobid=<id>

list volumes pool=<pool-name>

list volumes job=<job-name>

list volume=<volume-name>

list nextvolume job=<job-name>

list nextvol job=<job-name>

list nextvol job=<job-name> days=nnn
```

What most of the above commands do should be more or less obvious. In general if you do not specify all the command line arguments, the command will prompt you for what is needed.

The **list nextvol** command will print the Volume name to be used by the specified job. You should be aware that exactly what Volume will be used depends on a lot of factors including the time and what a prior job will do. It may fill a tape that is not full when you issue this command. As a consequence, this command will give you a good estimate of what Volume will be used but not a definitive answer. In addition, this command may have certain side effect because it runs through the same algorithm as a job, which means it may automatically purge or recycle a Volume. By default, the job specified must run within the next two days or no volume will be found. You can, however, use the **days=nnn** specification to specify up to 50 days. For example, if on Friday, you want to see what Volume will be needed on Monday, for job MyJob, you would use **list nextvol job=MyJob days=3**.

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. However, this takes some knowledge of programming SQL. Please see the **query** command below for additional information. See below for listing the full contents of a catalog record with the **llist** command.

As an example, the command **list pools** might produce the following output:

```
+------+---------+---------+---------+----------+-------------+
| PoId | Name    | NumVols | MaxVols | PoolType | LabelFormat |
+------+---------+---------+---------+----------+-------------+
|    1 | Default |       0 |       0 | Backup   | *           |
|    2 | Recycle |       0 |       8 | Backup   | File        |
+------+---------+---------+---------+----------+-------------+
```

As mentioned above, the **list** command lists what is in the database. Some things are put into the database immediately when Bacula starts up, but in general, most things are put in only when they are first used, which is the case for a Client as with Job records, etc.

Bacula should create a client record in the database the first time you run a job for that client. Doing a **status** will not cause a database record to be created. The client database record will be created whether or not the job fails, but it must at least start. When the Client is actually contacted, additional info from the client will be added to the client record (a "uname -a" output).

If you want to see what Client resources you have available in your conf file, you use the Console command **show clients**.

llist The llist or "long list" command takes all the same arguments that the list command described above does. The difference is that the llist command list the full contents of each database record selected. It does so by listing the various fields of the record vertically, with one field per line. It is possible to produce a very large number of output lines with this command.

If instead of the **list pools** as in the example above, you enter **llist pools** you might get the following output:

```
          PoolId: 1
            Name: Default
         NumVols: 0
         MaxVols: 0
         UseOnce: 0
      UseCatalog: 1
 AcceptAnyVolume: 1
    VolRetention: 1,296,000
  VolUseDuration: 86,400
      MaxVolJobs: 0
     MaxVolBytes: 0
```

```
        AutoPrune: 0
          Recycle: 1
         PoolType: Backup
      LabelFormat: *

           PoolId: 2
             Name: Recycle
          NumVols: 0
          MaxVols: 8
          UseOnce: 0
       UseCatalog: 1
  AcceptAnyVolume: 1
     VolRetention: 3,600
   VolUseDuration: 3,600
       MaxVolJobs: 1
      MaxVolBytes: 0
        AutoPrune: 0
          Recycle: 1
         PoolType: Backup
      LabelFormat: File
```

messages This command causes any pending console messages to be immediately displayed.

mount The mount command is used to get Bacula to read a volume on a physical device. It is a way to tell Bacula that you have mounted a tape and that Bacula should examine the tape. This command is normally used only after there was no Volume in a drive and Bacula requests you to mount a new Volume or when you have specifically unmounted a Volume with the **unmount** console command, which causes Bacula to close the drive. If you have an autoloader, the mount command will not cause Bacula to operate the autoloader unless you specify a **slot** and possibly a **drive**. The various forms of the mount command are:

mount storage=<storage-name> [slot=<num>] [drive=<num>]

mount [jobid=<id> — job=<job-name>]

If you have specified **Automatic Mount = yes** in the Storage daemon's Device resource, under most circumstances, Bacula will automatically access the Volume unless you have explicitly **unmount**ed it in the Console program.

python The python command takes a single argument **restart**:

python restart

This causes the Python interpreter in the Director to be reinitialized. This can be helpful for testing because once the Director starts and the Python interpreter is initialized, there is no other way to make it accept any changes to the startup script **DirStartUp.py**. For more details on Python scripting, please see the Python Scripting chapter of this manual.

prune The Prune command allows you to safely remove expired database records from Jobs and Volumes. This command works only on the Catalog database and does not affect data written to Volumes. In all cases, the Prune command applies a retention period to the specified records. You can Prune expired File entries from Job records; you can Prune expired Job records from the database, and you can Prune both expired Job and File records from specified Volumes.

prune files—jobs—volume client=<client-name> volume=<volume-name>

For a Volume to be pruned, the **VolStatus** must be Full, Used, or Append, otherwise the pruning will not take place.

purge The Purge command will delete associated Catalog database records from Jobs and Volumes without considering the retention period. **Purge** works only on the Catalog database and does not affect data written to Volumes. This command can be dangerous because you can delete catalog records associated with current backups of files, and we recommend that you do not use it unless you know what you are doing. The permitted forms of **purge** are:

purge files jobid=<jobid>—job=<job-name>—client=<client-name>

purge jobs client=<client-name> (of all jobs)

purge volume—volume=<vol-name> (of all jobs)

For the **purge** command to work on Volume Catalog database records the **VolStatus** must be Append, Full, Used, or Error.

The actual data written to the Volume will be unaffected by this command.

relabel This command is used to label physical volumes. The full form of this command is:

relabel storage=<storage-name> oldvolume=<old-volume-name> volume=<newvolume-name>

If you leave out any part, you will be prompted for it. In order for the Volume (old-volume-name) to be relabeled, it must be in the catalog, and the volume status must be marked **Purged** or **Recycle**. This happens automatically as a result of applying retention periods, or you may explicitly purge the volume using the **purge** command.

Once the volume is physically relabeled, the old data previously written on the Volume is lost and cannot be recovered.

release This command is used to cause the Storage daemon to rewind (release) the current tape in the drive, and to re-read the Volume label the next time the tape is used.

release storage=<storage-name>

After a release command, the device is still kept open by Bacula (unless Always Open is set to No in the Storage Daemon's configuration) so it cannot be used by another program. However, with some tape drives, the operator can remove the current tape and to insert a different one, and when the next Job starts, Bacula will know to re-read the tape label to find out what tape is mounted. If you want to be able to use the drive with another program (e.g. **mt**), you must use the **unmount** command to cause Bacula to completely release (close) the device.

reload The reload command causes the Director to re-read its configuration file and apply the new values. The new values will take effect immediately for all new jobs. However, if you change schedules, be aware that the scheduler pre-schedules jobs up to two hours in advance, so any changes that are to take place during the next two hours may be delayed. Jobs that have already been scheduled to run (i.e. surpassed their requested start time) will continue with the old values. New jobs will use the new values. Each time you issue a reload command while jobs are running, the prior config values will queued until all jobs that were running before issuing the reload terminate, at which time the old config values will be released from memory. The Directory permits keeping up to ten prior set of configurations before it will refuse a reload command. Once at least one old set of config values has been released it will again accept new reload commands.

While it is possible to reload the Director's configuration on the fly, even while jobs are executing, this is a complex operation and not without side effects. Accordingly, if you have to reload the Director's configuration while Bacula is running, it is advisable to restart the Director at the next convenient opportunity.

restore The restore command allows you to select one or more Jobs (JobIds) to be restored using various methods. Once 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.

restore storage=<storage-name> client=<backup-client-name> where=<path> pool=<pool-name> fileset=<fileset-name> restoreclient=<restore-client-name> select current all done

Where **current**, if specified, tells the restore command to automatically select a restore to the most current backup. If not specified, you will be prompted. The **all** specification tells the restore command to restore all files. If it is not specified, you will be prompted for the files to restore. For details of the **restore** command, please see the Restore Chapter of this manual.

The client keyword initially specifies the client from which the backup was made and the client to which the restore will be make. However, if the restoreclient keyword is specified, then the restore is written to that client.

run This command allows you to schedule jobs to be run immediately. The full form of the command is:

run job=<job-name> client=<client-name> fileset=<FileSet-name> level=<level-keyword> storage=<storage-name> where=<directory-prefix> when=<universal-time-specification> yes

Any information that is needed but not specified will be listed for selection, and before starting the job, you will be prompted to accept, reject, or modify the parameters of the job to be run, unless you have specified **yes**, in which case the job will be immediately sent to the scheduler.

On my system, when I enter a run command, I get the following prompt:

```
A job name must be specified.
The defined Job resources are:
     1: Matou
     2: Polymatou
     3: Rufus
     4: Minimatou
     5: Minou
     6: PmatouVerify
     7: MatouVerify
     8: RufusVerify
     9: Watchdog
Select Job resource (1-9):
```

If I then select number 5, I am prompted with:

```
Run Backup job
JobName:  Minou
```

```
FileSet:  Minou Full Set
Level:    Incremental
Client:   Minou
Storage:  DLTDrive
Pool:     Default
When:     2003-04-23 17:08:18
OK to run? (yes/mod/no):
```

If I now enter **yes**, the Job will be run. If I enter **mod**, I will be presented with the following prompt.

```
Parameters to modify:
     1: Level
     2: Storage
     3: Job
     4: FileSet
     5: Client
     6: When
     7: Pool
Select parameter to modify (1-7):
```

If you wish to start a job at a later time, you can do so by setting the When time. Use the **mod** option and select **When** (no. 6). Then enter the desired start time in YYYY-MM-DD HH:MM:SS format.

setdebug This command is used to set the debug level in each daemon. The form of this command is:

setdebug level=nn [trace=0/1 client=<client-name> — dir — director — storage=<storage-name> — all]

If trace=1 is set, then tracing will be enabled, and the daemon will be placed in trace mode, which means that all debug output as set by the debug level will be directed to the file **bacula.trace** in the current directory of the daemon. Normally, tracing is needed only for Win32 clients where the debug output cannot be written to a terminal or redirected to a file. When tracing, each debug output message is appended to the trace file. You must explicitly delete the file when you are done.

show The show command will list the Director's resource records as defined in the Director's configuration file (normally **bacula-dir.conf**). This command is used mainly for debugging purposes by developers. The following keywords are accepted on the show command line: catalogs, clients, counters, devices, directors, filesets, jobs, messages, pools, schedules, storages, all, help. Please don't confuse this command with the **list**, which displays the contents of the catalog.

sqlquery The sqlquery command puts the Console program into SQL query mode where each line you enter is concatenated to the previous line until a semicolon (;) is seen. The semicolon terminates the command, which is then passed directly to the SQL database engine. When the output from the SQL engine is displayed, the formation of a new SQL command begins. To terminate SQL query mode and return to the Console command prompt, you enter a period (.) in column 1.

Using this command, you can query the SQL catalog database directly. Note you should really know what you are doing otherwise you could damage the catalog database. See the **query** command below for simpler and safer way of entering SQL queries.

Depending on what database engine you are using (MySQL, PostgreSQL or SQLite), you will have somewhat different SQL commands available. For more detailed information, please refer to the MySQL, PostgreSQL or SQLite documentation.

status This command will display the status of the next jobs that are scheduled during the next 24 hours as well as the status of currently running jobs. The full form of this command is:

status [all — dir=<dir-name> — director — client=<client-name> — storage=<storage-name> — days=nnn]

If you do a **status dir**, the console will list any currently running jobs, a summary of all jobs scheduled to be run in the next 24 hours, and a listing of the last ten terminated jobs with their statuses. The scheduled jobs summary will include the Volume name to be used. You should be aware of two things: 1. to obtain the volume name, the code goes through the same code that will be used when the job runs, but it does not do pruning nor recycling of Volumes; 2. The Volume listed is at best a guess. The Volume actually used may be different because of the time difference (more durations may expire when the job runs) and another job could completely fill the Volume requiring a new one.

In the Running Jobs listing, you may find the following types of information:

```
2507 Catalog MatouVerify.2004-03-13_05.05.02 is waiting execution
5349 Full    CatalogBackup.2004-03-13_01.10.00 is waiting for higher
             priority jobs to finish
5348 Differe Minou.2004-03-13_01.05.09 is waiting on max Storage jobs
5343 Full    Rufus.2004-03-13_01.05.04 is running
```

Looking at the above listing from bottom to top, obviously JobId 5343 (Rufus) is running. JobId 5348 (Minou) is waiting for JobId 5343 to finish because it is using the Storage resource, hence the "waiting on max Storage jobs". JobId 5349 has a lower priority than all the other jobs so it is waiting for higher priority jobs to finish, and finally, JobId 2508 (MatouVerify) is waiting because only one job can run at a time, hence it is simply "waiting execution"

If you do a **status dir**, it will by default list the first occurrence of all jobs that are scheduled today and tomorrow. If you wish to see the jobs that are scheduled in the next three days (e.g. on Friday you want to see the first occurrence of what tapes are scheduled to be used on Friday, the weekend, and Monday), you can add the **days=3** option. Note, a **days=0** shows the first occurrence of jobs scheduled today only. If you have multiple run statements, the first occurrence of each run statement for the job will be displayed for the period specified.

If your job seems to be blocked, you can get a general idea of the problem by doing a **status dir**, but you can most often get a much more specific indication of the problem by doing a **status storage=xxx**. For example, on an idle test system, when I do **status storage=File**, I get:

```
status storage=File
Connecting to Storage daemon File at 192.168.68.112:8103

rufus-sd Version: 1.39.6 (24 March 2006) i686-pc-linux-gnu redhat (Stentz)
Daemon started 26-Mar-06 11:06, 0 Jobs run since started.

Running Jobs:
No Jobs running.
====

Jobs waiting to reserve a drive:
====

Terminated Jobs:
 JobId  Level   Files          Bytes Status   Finished        Name
======================================================================
    59  Full        234      4,417,599 OK       15-Jan-06 11:54 kernsave
====

Device status:
utochanger "DDS-4-changer" with devices:
   "DDS-4" (/dev/nst0)
Device "DDS-4" (/dev/nst0) is mounted with Volume="TestVolume002"
Pool="*unknown*"
    Slot 2 is loaded in drive 0.
    Total Bytes Read=0 Blocks Read=0 Bytes/block=0
    Positioned at File=0 Block=0
Device "Dummy" is not open or does not exist.
No DEVICE structure.

Device "DVD-Writer" (/dev/hdc) is not open.
```

```
Device "File" (/tmp) is not open.
====

In Use Volume status:
====
```

Now, what this tells me is that no jobs are running and that none of the devices are in use. Now, if I **unmount** the autochanger, which will not be used in this example, and then start a Job that uses the File device, the job will block. When I re-issue the status storage command, I get for the Device status:

```
status storage=File
...
Device status:
Autochanger "DDS-4-changer" with devices:
   "DDS-4" (/dev/nst0)
Device "DDS-4" (/dev/nst0) is not open.
    Device is BLOCKED. User unmounted.
    Drive 0 is not loaded.
Device "Dummy" is not open or does not exist.
No DEVICE structure.

Device "DVD-Writer" (/dev/hdc) is not open.
Device "File" (/tmp) is not open.
    Device is BLOCKED waiting for media.
====
...
```

Now, here it should be clear that if a job were running that wanted to use the Autochanger (with two devices), it would block because the user unmounted the device. The real problem for the Job I started using the "File" device is that the device is blocked waiting for media – that is Bacula needs you to label a Volume.

unmount This command causes the indicated Bacula Storage daemon to unmount the specified device. The forms of the command are the same as the mount command:

```
unmount storage=<storage-name> [ drive=<num> ]

unmount [ jobid=<id> | job=<job-name> ]
```

Once you unmount a storage device, Bacula will no longer be able to use it until you issue a mount command for that device. If Bacula needs to access that device, it will block and issue mount requests periodically to the operator.

If the device you are unmounting is an autochanger, it will unload the drive you have specified on the command line. If no drive is specified, it will assume drive 1.

update This command will update the catalog for either a specific Pool record, a Volume record, or the Slots in an autochanger with barcode capability. In the case of updating a Pool record, the new information will be automatically taken from the corresponding Director's configuration resource record. It can be used to increase the maximum number of volumes permitted or to set a maximum number of volumes. The following main keywords may be specified:

```
media, volume, pool, slots
```

In the case of updating a Volume, you will be prompted for which value you wish to change. The following Volume parameters may be changed:

```
Volume Status
Volume Retention Period
Volume Use Duration
Maximum Volume Jobs
Maximum Volume Files
Maximum Volume Bytes
Recycle Flag
Recycle Pool
Slot
InChanger Flag
Pool
Volume Files
Volume from Pool
All Volumes from Pool
All Volumes from all Pools
```

For slots **update slots**, Bacula will obtain a list of slots and their barcodes from the Storage daemon, and for each barcode found, it will automatically update the slot in the catalog Media record to correspond to the new value. This is very useful if you have moved cassettes in the magazine, or if you have removed the magazine and inserted a different one. As the slot of each Volume is updated, the InChanger flag for that Volume will also be set, and any other Volumes in the Pool that were last mounted on the same Storage device will have their InChanger flag turned off. This permits Bacula to know what magazine (tape holder) is currently in the autochanger.

If you do not have barcodes, you can accomplish the same thing in version 1.33 and later by using the **update slots scan** command. The **scan** keyword tells Bacula to physically mount each tape and to read its VolumeName.

For Pool **update pool**, Bacula will move the Volume record from its existing pool to the pool specified.

For **Volume from Pool**, **All Volumes from Pool** and **All Volumes from all Pools**, the following values are updated from the Pool record: Recycle, RecyclePool, VolRetention, VolUseDuration, MaxVolJobs, MaxVolFiles, and MaxVolBytes. (RecyclePool feature is available with bacula 2.1.4 or higher.)

The full form of the update command with all command line arguments is:

```
update volume=xxx pool=yyy slots volstatus=xxx VolRetention=ddd
  VolUse=ddd MaxVolJobs=nnn MaxVolBytes=nnn Recycle=yes|no
  slot=nnn enabled=n recyclepool=zzz
```

use This command allows you to specify which Catalog database to use. Normally, you will be using only one database so this will be done automatically. In the case that you are using more than one database, you can use this command to switch from one to another.

use <database-name>

var This command takes a string or quoted string and does variable expansion on it the same way variable expansion is done on the **LabelFormat** string. Thus, for the most part, you can test your LabelFormat strings. The difference between the **var** command and the actual LabelFormat process is that during the var command, no job is running so "dummy" values are used in place of Job specific variables. Generally, however, you will get a good idea of what is going to happen in the real case.

version The command prints the Director's version.

quit This command terminates the console program. The console program sends the **quit** request to the Director and waits for acknowledgment. If the Director is busy doing a previous command for you that has not terminated, it may take some time. You may quit immediately by issuing the **.quit** command (i.e. quit preceded by a period).

query This command reads a predefined SQL query from the query file (the name and location of the query file is defined with the QueryFile resource record in the Director's configuration file). You are prompted to select a query from the file, and possibly enter one or more parameters, then the command is submitted to the Catalog database SQL engine.

The following queries are currently available (version 1.24):

```
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 20 Full Backups for a Client:
  7: List Volumes used by selected JobId:
  8: List Volumes to Restore All Files:
  9: List where a File is saved:
Choose a query (1-9):
```

exit This command terminates the console program.

wait The wait command causes the Director to pause until there are no jobs running. This command is useful in a batch situation such as regression testing where you wish to start a job and wait until that job completes before continuing. This command now has the following options:

```
wait [jobid=nn] [jobuid=unique id] [job=job name]
```

If specified with a specific JobId, ... the wait command will wait for that particular job to terminate before continuing.

18.6 Special dot Commands

There is a list of commands that are prefixed with a period (.). These commands are intended to be used either by batch programs or graphical user interface front-ends. They are not normally used by interactive users. Once GUI development begins, this list will be considerably expanded. The following is the list of dot commands:

```
.backups job=xxx      list backups for specified job
.clients              list all client names
.defaults client=xxx fileset=yyy  list defaults for specified client
.die                  cause the Director to segment fault (for debugging)
.dir                  when in tree mode prints the equivalent to the dir command,
                        but with fields separated by commas rather than spaces.
.exit                 quit
.filesets             list all fileset names
.help                 help command output
.jobs                 list all job names
.levels               list all levels
.messages             get quick messages
```

```
.msgs               return any queued messages
.pools              list all pool names
.quit               quit
.status             get status output
.storage            return storage resource names
.types              list job types
```

18.7 Special At (@) Commands

Normally, all commands entered to the Console program are immediately forwarded to the Director, which may be on another machine, to be executed. However, there is a small list of **at** commands, all beginning with an at character (@), that will not be sent to the Director, but rather interpreted by the Console program directly. Note, these commands are implemented only in the tty console program and not in the GNOME Console. These commands are:

@input <filename> Read and execute the commands contained in the file specified.

@output <filename> w/a Send all following output to the filename specified either overwriting the file (w) or appending to the file (a). To redirect the output to the terminal, simply enter **@output** without a filename specification. WARNING: be careful not to overwrite a valid file. A typical example during a regression test might be:

```
@output /dev/null
commands ...
@output
```

@tee <filename> w/a Send all subsequent output to both the specified file and the terminal. It is turned off by specifying **@tee** or **@output** without a filename.

@sleep <seconds> Sleep the specified number of seconds.

@time Print the current time and date.

@version Print the console's version.

@quit quit

@exit quit

@# anything Comment

18.8 Running the Console from a Shell Script

You can automate many Console tasks by running the console program from a shell script. For example, if you have created a file containing the following commands:

```
./bconsole -c ./bconsole.conf <<END_OF_DATA
unmount storage=DDS-4
quit
END_OF_DATA
```

when that file is executed, it will unmount the current DDS-4 storage device. You might want to run this command during a Job by using the **RunBeforeJob** or **RunAfterJob** records.

It is also possible to run the Console program from file input where the file contains the commands as follows:

```
./bconsole -c ./bconsole.conf <filename
```

where the file named **filename** contains any set of console commands.

As a real example, the following script is part of the Bacula regression tests. It labels a volume (a disk volume), runs a backup, then does a restore of the files saved.

```
bin/bconsole -c bin/bconsole.conf <<END_OF_DATA
@output /dev/null
messages
@output /tmp/log1.out
label volume=TestVolume001
run job=Client1 yes
wait
messages
@#
@# now do a restore
@#
@output /tmp/log2.out
restore current all
yes
wait
messages
@output
quit
END_OF_DATA
```

The output from the backup is directed to /tmp/log1.out and the output from the restore is directed to /tmp/log2.out. To ensure that the backup and restore ran correctly, the output files are checked with:

```
grep "^Termination: *Backup OK" /tmp/log1.out
backupstat=$?
grep "^Termination: *Restore OK" /tmp/log2.out
restorestat=$?
```

18.9 Adding Volumes to a Pool

If you have used the **label** command to label a Volume, it will be automatically added to the Pool, and you will not need to add any media to the pool.

Alternatively, you may choose to add a number of Volumes to the pool without labeling them. At a later time when the Volume is requested by **Bacula** you will need to label it.

Before adding a volume, you must know the following information:

1. The name of the Pool (normally "Default")
2. The Media Type as specified in the Storage Resource in the Director's configuration file (e.g. "DLT8000")
3. The number and names of the Volumes you wish to create.

For example, to add media to a Pool, you would issue the following commands to the console program:

```
*add
Enter name of Pool to add Volumes to: Default
Enter the Media Type: DLT8000
Enter number of Media volumes to create. Max=1000: 10
Enter base volume name: Save
Enter the starting number: 1
10 Volumes created in pool Default
*
```

To see what you have added, enter:

```
*list media pool=Default
+-------+----------+---------+---------+-------+------------------+
| MedId | VolumeNa | MediaTyp| VolStat | Bytes | LastWritten      |
+-------+----------+---------+---------+-------+------------------+
|    11 | Save0001 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    12 | Save0002 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    13 | Save0003 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    14 | Save0004 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    15 | Save0005 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    16 | Save0006 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    17 | Save0007 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    18 | Save0008 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    19 | Save0009 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    20 | Save0010 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
+-------+----------+---------+---------+-------+------------------+
*
```

Notice that the console program automatically appended a number to the base Volume name that you specify (Save in this case). If you don't want it to append a number, you can simply answer 0 (zero) to the question "Enter number of Media volumes to create. Max=1000:", and in this case, it will create a single Volume with the exact name you specify.

Chapter 19

The Restore Command

19.1 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 Bacula Utility Programs chapter of this manual. 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 Bacula Utility Programs chapter.

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.

19.2 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. See 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: Cancel
Select item:  (1-12):
```

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 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
 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**.

19.3 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.

19.4 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, and the **yes** suppresses the final **yes/mod/no** prompt and simply runs the restore.

The full list of possible command line arguments are:

- **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** – select the specified 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.
- **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-restore** – restore files in **where** directory.
- **yes** – automatically run the restore without prompting for modifications (most useful in batch scripts).
- **strip_prefix=/prod** – remove a part of the filename when restoring.
- **add_prefix=/test** – add a prefix to all files when restoring (like where) (can't be used with **where=**).
- **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.

19.5 Using File Relocation

19.5.1 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. In this case, you could use **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):
```

19.5.2 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

Orignal filename	Computed filename	RegexWhere	Comments
`c:/system.ini`	`c:/system.old.ini`	`/.ini$/.old.ini/`	use $ as enc
`/prod/u01/pdata/`	`/rect/u01/rdata`	`/prod/rect/,/pdata/rdata/`	using two re
`/prod/u01/pdata/`	`/rect/u01/rdata`	`!/prod/!/rect/!,/pdata/rdata/`	using ! inst
`C:/WINNT`	`d:/WINNT`	`/c:/d:/i`	using case-i

19.6 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), nor will it restore non-portable Win32 data (default) to Unix machines.

19.7 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.

19.8 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).

19.9 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.

19.10 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.

19.11 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.

19.12 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.

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.

19.13 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 SQLite, use the vacuum command to try to fix the database. 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 dbcheck 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?

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 bf 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 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 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 of the Volume Utility Tools of this chapter for more details.

Chapter 20

GUI Programs

20.1 List of GUI Programs

This document briefly describes the GUI programs that work with Bacula. The GUI programs that are currently available are:

bat **bat** is short for Bacula Administration Tool. It is a GUI form of bconsole, but with many additional features. Although we are still working on adding new features to bat, at this point, it has more features than both bwx-console and the bgnome-console, and over time many additional features will be added, including the functionality of the tray-monitor as well as the reporting capabilities of bweb.

It is very difficult to provide a guide for using bat other than to say to try it. In many of the graphical display "panes" (shown in the right window), you can click with the right mouse button to bring up a context sensitive menu that provides quite a lot of features that can be easily missed.

The Bacula wiki has a number of screenshots of bat.

To build bat, you will need to have Qt4 ¿= 4.2 loaded (libraries and the devel libraries) as well as the QWT graphics package. Please see the enable-bat section of the Installation chapter of this manual for the details of how to build it.

The major bat features are:

- Graphical features
  - Undockable (floating) windows of all interfaces.
  - Page selector for deciding and finding interfaces to view.

  - Console command window for entering standard text console commands.
  - A preferences interface for listing limits and debugging.
- Graphical console
  - A console to perform all the commands available from bconsole.
- Graphical restoring
  - High performance restore of backup with GUI tree browsing and file and directory selecting. Pre-restore Interface to assist in selecting jobs for this restore method.
  - Restore by browsing and selecting from all cataloged versions of files. This method allows for Multiple simultaneous views of catalog data. The only selection limitation for this browsing method is one client at a time.
- Graphical listing Interfaces
  - List the backup jobs that have run. ** see details below
  - List the clients and perform 4 commands on the client within context.
  - List the Filesets
  - List the Job resources and perform 8 commands on the job within context.
  - List the Pools and the volumes in each pool. Perform 7 different commands on each media.
  - List the storage resources and perform any of 5 commands on the storage resource.
- Graphical Media Management
  - Modify Volume parameters
  - Label a new volume
  - Relabel an existing volume
  - Select jobs on a volume in Job List directly from media interface
  - Delete a Volume
  - Purge Jobs on volume
  - Update Volume Parameters from Pool Parameters
- Graphical Graphing
  - Interface to plot the files and bytes of backup jobs.
- Other Graphical interfaces:
  - Open the cataloged Log messages of a job that has run
  - Run a job manually and modify job defaults before running.

  - Estimate a job. Determine the files and bytes that would be backed up if the job were run now.
- JobList features
  - As many joblist windows at a time as desired.
  - Nine different selection criterion: client, volume, job, Fileset, level, status, purged or not, record limit and days limit
  - Lists the 11 most commonly desired job attributes.
  - Run the following console commands by issuing the command within the context of the know job number: List job, list files on job, list jobmedia, list volumes, delete job and purge files.
  - Open other interfaces knowing the jobid: show the cataloged log of the running of the job, restore from that job only browsing the filestructure, restore from the job end time to get the most recent version of a restore after the job completed running and browse the combined filestucture, jump directly to view a plot of the selected jobs files and bytes backed up.

Bat also has a nice online help manual that explains a lot of the interface.

bimagemgr Bimagemgr is a web based interface written in Perl that monitors disk Volumes intended to be written to CDROM.

For more information on bimagemgr, please see below.

bwx-console bwx-console is a graphical console interface written in wxWidgets and available on all client platforms. bwx-console allows you to do anything you can do in the standard tty console and in addition has a graphic tree based point and click restore feature.

bgnome-console The bgnome-console is a graphical console interface available on systems that support GNOME 2.x. Although it runs in its own graphical window and permits all the standard console commands, it has almost no additional graphical features implemented.

For more information on bgnome-console, please consult the Console Chapter of this manual.

tray-monitor The tray-monitor is a daemon monitoring program that resides in the system tray on GNOME and KDE systems. It is a monitor program that will show you the status of any daemon. It is not a program for interfacing to the console.

For more information, please see Configuring the Monitor Program chapter this manual.

bweb Bweb is a perl based web program that provides a tool to do basic operations and get statistics. (it requires Bacula ¿= 1.39) It obtains its information from your catalog database and the bconsole program.

Some of its major features are the following:

1. Follow, in real time, job progression (with client status and job log)
2. See Pool/Media occupation
3. Update volume parameters
4. Manage locations (with a workflow to move in/out media)
5. Get statistics about jobs (file number, job size, job duration)
6. Run a new job
7. Manage your autochanger (put cartridge on I/O, empty I/O slots with free slots, etc.)
8. Works with both PostgreSQL and MySQL
9. Works correctly under Mozilla and Firefox

Please, read the INSTALL file in the bweb source directory for detailed instructions on getting it to work.

brestore Brestore is a graphical restoration interface available on systems that support Perl/GTK/Glade. (it requires Bacula ¿= 1.38) It has the following features:

1. Direct SQL access to the database for good performance
2. Fast Time Navigation (switch almost instantaneously between the different versions of a directory, by changing the date from a list)
3. Possibility to choose a selected file, then browse all its available versions, and directly see if these versions are online in a library or not
4. Simple restoration by generation of a BSR file
5. Works with both PostgreSQL and MySQL
6. Works with bweb to follow job

Please, read README file in the bweb source directory for detailed instructions on getting it to work.

bacula-web Bacula-web is a php based web program that provides a summarized output of jobs that have already run. It obtains its information from your catalog database. Aside from a nice graphical display, it provides summaries of your jobs, as well as graphs of job usage. This is a fairly high level bacula management tool.

Here are a few points that one user made concerning this tool:

1. It is web-based so can be accessed from anywhere.
2. It is "read only". Users can examine the state of the backups but cannot write to anything and therefore can do no damage.
3. It packs a phenomenal amount of information into a single web-page - that I credit as being very good design!

The documentation for bacula-web can be found in a separate bacula-web document that in the **bacula-docs** release.

20.2 bimagemgr

bimagemgr is a utility for those who backup to disk volumes in order to commit them to CDR disk, rather than tapes. It is a web based interface written in Perl and is used to monitor when a volume file needs to be burned to disk. It requires:

- A web server running on the bacula server
- A CD recorder installed and configured on the bacula server
- The cdrtools package installed on the bacula server.
- perl, perl-DBI module, and either DBD-MySQL DBD-SQLite or DBD-PostgreSQL modules

DVD burning is not supported by **bimagemgr** at this time, but both are planned for future releases.

20.2.1 bimagemgr installation

Installation from tarball: 1. Examine the Makefile and adjust it to your configuration if needed. 2. Edit config.pm to fit your configuration. 3. Do 'make install' as root. 4. Edit httpd.conf and change the Timeout value. The web server must not time out and close the connection before the burn process is finished. The exact value needed may vary depending upon your cd recorder speed and whether you are burning on the bacula server on on another machine across your network. In my case I set it to 1000 seconds. Restart httpd. 5. Make sure that cdrecord is setuid root.

Installation from rpm package:

1. Install the rpm package for your platform.
2. Edit /cgi-bin/config.pm to fit your configuration.
3. Edit httpd.conf and change the Timeout value. The web server must not time out and close the connection before the burn process is finished. The exact value needed may vary depending upon your cd recorder speed and whether you are burning on the bacula server on on another machine across your network. In my case I set it to 1000 seconds. Restart httpd.

4. Make sure that cdrecord is setuid root.

For bacula systems less than 1.36:

1. Edit the configuration section of config.pm to fit your configuration.

2. Run /etc/bacula/create_cdimage_table.pl from a console on your bacula server (as root) to add the CDImage table to your bacula database.

Accessing the Volume files: The Volume files by default have permissions 640 and can only be read by root. The recommended approach to this is as follows (and only works if bimagemgr and apache are running on the same host as bacula.

For bacula-1.34 or 1.36 installed from tarball -

1. Create a new user group bacula and add the user apache to the group for Red Hat or Mandrake systems. For SuSE systems add the user wwwrun to the bacula group.

2. Change ownership of all of your Volume files to root.bacula

3. Edit the /etc/bacula/bacula startup script and set SD_USER=root and SD_GROUP=bacula. Restart bacula.

Note: step 3 should also be done in /etc/init.d/bacula-sd but released versions of this file prior to 1.36 do not support it. In that case it would be necessary after a reboot of the server to execute '/etc/bacula/bacula restart'.

For bacula-1.38 installed from tarball - 1. Your configure statement should include:

```
--with-dir-user=bacula
--with-dir-group=bacula
--with-sd-user=bacula
--with-sd-group=disk
--with-fd-user=root
--with-fd-group=bacula
```

2. Add the user apache to the bacula group for Red Hat or Mandrake systems. For SuSE systems add the user wwwrun to the bacula group.

3. Check/change ownership of all of your Volume files to root.bacula

For bacula-1.36 or bacula-1.38 installed from rpm -

1. Add the user apache to the group bacula for Red Hat or Mandrake systems. For SuSE systems add the user wwwrun to the bacula group.

2. Check/change ownership of all of your Volume files to root.bacula

bimagemgr installed from rpm ¿ 1.38.9 will add the web server user to the bacula group in a post install script. Be sure to edit the configuration information in config.pm after installation of rpm package.

bimagemgr will now be able to read the Volume files but they are still not world readable.

If you are running bimagemgr on another host (not recommended) then you will need to change the permissions on all of your backup volume files to 644 in order to access them via nfs share or other means. This approach should only be taken if you are sure of the security of your environment as it exposes the backup Volume files to world read.

20.2.2 bimagemgr usage

Calling the program in your web browser, e.g. `http://localhost/cgi-bin/bimagemgr.pl` will produce a display as shown below in Figure 1. The program will query the bacula database and display all volume files with the date last written and the date last burned to disk. If a volume needs to be burned (last written is newer than last burn date) a "Burn" button will be displayed in the rightmost column.

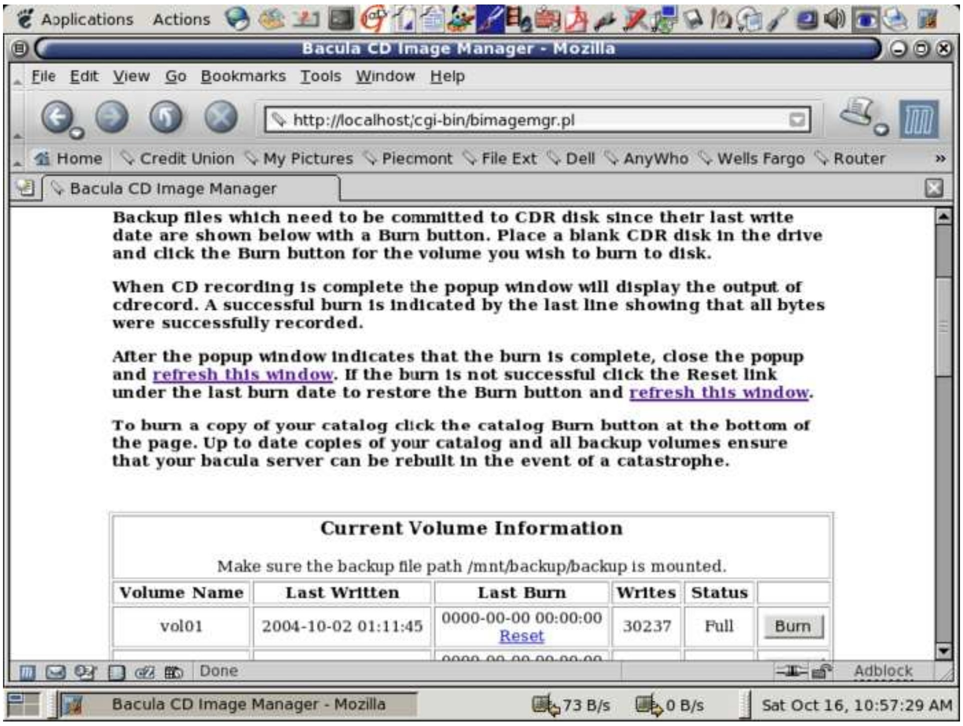


Figure 1

Place a blank CDR disk in your recorder and click the "Burn" button. This will cause a pop up window as shown in Figure 2 to display the burn progress.

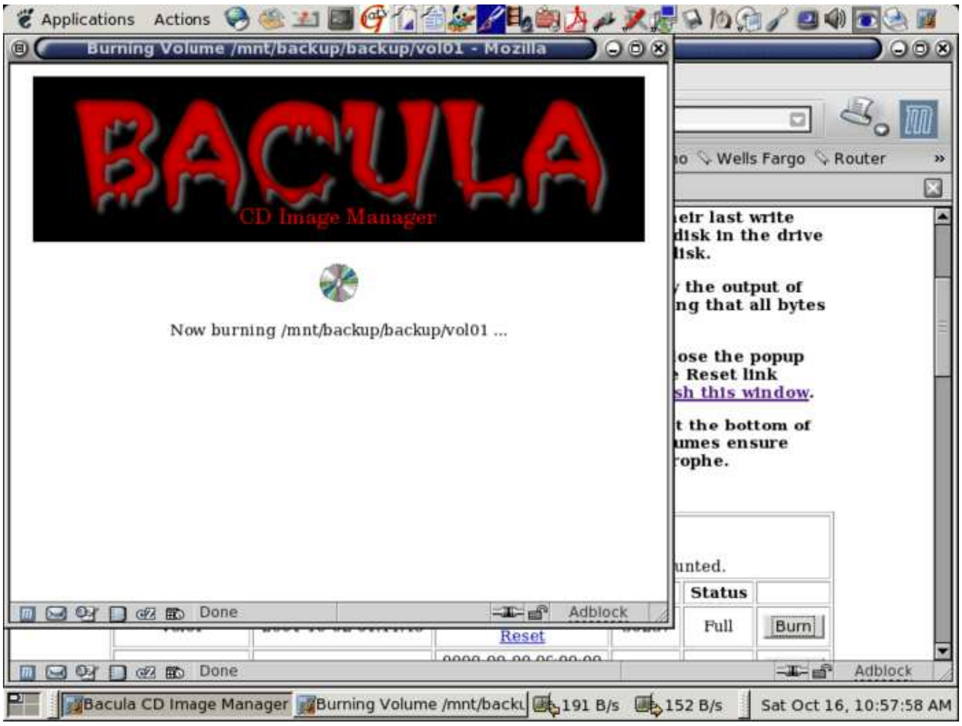


Figure 2

When the burn finishes the pop up window will display the results of cdrecord as shown in Figure 3. Close the pop up window and refresh the main window. The last burn date will be updated and the "Burn" button for that volume will disappear. Should you have a failed burn you can reset the last burn date of that volume by clicking its "Reset" link.

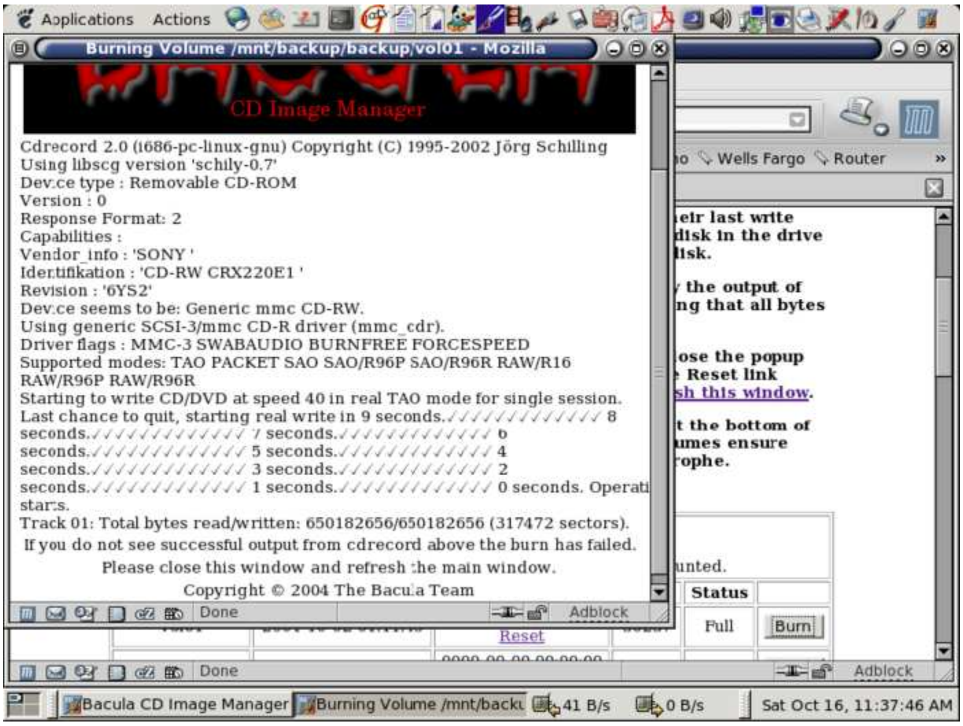


Figure 3

In the bottom row of the main display window are two more buttons labeled "Burn Catalog" and "Blank CDRW". "Burn Catalog" will place a copy of your bacula catalog on a disk. If you use CDRW disks rather than CDR then "Blank CDRW" allows you to erase the disk before re-burning it. Regularly committing your backup volume files and your catalog to disk with **bimagemgr** ensures that you can rebuild easily in the event of some disaster on the bacula server itself.

Chapter 21

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.

21.1 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.

21.2 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, and SQLite version 2.8.4 and greater has the **VACUUM** command. 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 and SQLite.

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.

21.3 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: http://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
```

21.4 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: http://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.

21.5 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:

http://dev.mysql.com/doc/refman/5.0/en/gone-away.html

21.6 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).

21.7 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 SQLite3, one significant factor in improving the performance is to ensure that there is a "PRAGMA synchronous = NORMAL;" statement. This reduces the number of times that the database flushes the in memory cache to disk. There are other settings for this PRAGMA that can give even further performance improvements at the risk of a database corruption if your system crashes.

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: http://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: http://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.

21.8 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 three 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 two are the JobId index and the (Filename, PathId) index. If these Indexes are not present, your performance may suffer a lot.

21.8.1 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_fp_idx on file (filenameid, pathid);
```

21.8.2 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, FilenameId, PathId);
```

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 Filename;
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);

DROP INDEX Name on Filename;
CREATE INDEX Name on Filename (Name(255));
```

21.8.3 SQLite Indexes

On SQLite, you can check if you have the proper indexes by:

```
sqlite <path>bacula.db
select * from sqlite_master where type='index' and tbl_name='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_jfp_idx on File (JobId, FilenameId, PathId);
```

21.9 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 http://www.postgresql.org/docs/8.1/interactive/maintenance.html.

21.10 Compacting Your SQLite Database

First please read the previous section that explains why it is necessary to compress a database. SQLite version 2.8.4 and greater have the **Vacuum** command for compacting the database.

```
cd {\bf working-directory}
echo 'vacuum;' | sqlite bacula.db
```

As an alternative, you can use the following commands, adapted to your system:

```
cd {\bf working-directory}
echo '.dump' | sqlite bacula.db > bacula.sql
rm -f bacula.db
sqlite bacula.db < bacula.sql
rm -f bacula.sql
```

Where **working-directory** is the directory that you specified in the Director's configuration file. Note, in the case of SQLite, it is necessary to completely delete (rm) the old database before creating a new compressed version.

21.11 Migrating from SQLite to MySQL or PostgreSQL

You may begin using Bacula with SQLite then later find that you want to switch to MySQL or Postgres for any of a number of reasons: 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 a simple process. Scripts are available on bacula source distribution under `examples/database`.

21.12 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 instable 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** and **delete_catalog_backup**. 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 for details.
  RunBeforeJob = "/home/kern/bacula/bin/make_catalog_backup"
  RunAfterJob  = "/home/kern/bacula/bin/delete_catalog_backup"
  Write Bootstrap = "/home/kern/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 = \lt{}working_directory\gt{}/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.

21.13 Security considerations

We provide make_catalog_backup 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 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 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, and we know MySQL has .my.cnf.

Only you can decide what is appropriate for your situation. We have provided you with a starting point. We hope it helps.

21.14 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.

21.15 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 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.

If we were using SQLite, the determination of the total database size would be much easier since it is a single file, but we would have less insight to the size of the individual tables as we have in this case.

Note, SQLite databases may be as much as 50% larger than MySQL databases due to the fact that all data is stored as ASCII strings. That is even binary integers are stored as ASCII strings, and this seems to increase the space needed.

Chapter 22

Automatic Volume Recycling

By default, once Bacula starts writing a Volume, it can 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. Thus all previous data 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) tapes. This would require a large number of tapes though, and in such a case, it is possible to manually recycle tapes. For more on manual recycling, see the section entitled Manually Recycling Volumes below in this chapter.

Most people prefer to have a Pool of tapes that are used for daily backups and recycled once a week, another Pool of tapes that are used for Full backups once a week and recycled monthly, and finally a Pool of tapes that are used once a month and recycled after a year or two. With a scheme like this, the number of tapes 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> (*This require bacula 2.1.4 or greater*)

The above three directives are all you need assuming that you fill each of your Volumes then wait the Volume Retention period before reusing them. 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 whose Volume Status is **Full**. At that point, the recycling occurs in two steps. The first is that the Catalog for a Volume must be purged of all Jobs and Files contained on that Volume, and the second step is the actual recycling of the Volume. The Volume will be purged if the VolumeRetention period has expired. When a Volume is marked as Purged, it means that no Catalog records reference that Volume, and the Volume can be recycled. 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 Volume 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 tapes (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.

22.1 Automatic Pruning

As Bacula writes files to tape, 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 an entry reaches a certain age (e.g. 30 days old) it is removed from the catalog. Once a job has been pruned, you can still restore it from the backup tape, 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 tape. 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 in the console. (thanks to Bryce Denney for the above explanation).

22.2 Pruning Directives

There are three pruning durations. All apply to catalog database records and not to the actual data in a Volume. The pruning (or retention) durations are for: Volumes (Media records), 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 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 the 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 all Volumes in the Pool when a Job needs another Volume. Volume pruning means removing 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 another volume and there are no Volumes with Volume Status **Append** 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 counting from the time the last job stored on the Volume terminated. A key point is that this time period is not even considered as long at the Volume remains appendable. The Volume Retention period count down begins only when the Append status has been changed to some othe status (Full, Used, Purged, ...).

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.

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, 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 file 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 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 tape 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.

22.3 Recycling Algorithm

After all Volumes of a Pool have been pruned (as mentioned above, this happens when a Job needs a new Volume and no appendable Volumes are available), Bacula will look for the oldest Volume that is Purged (all Jobs and Files expired), 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).
- Try recycling any purged Volumes.
- Prune volumes applying Volume retention period (Volumes with VolStatus Full, Used, or Append are pruned). Note, even if all the File and Job records are pruned from a Volume, the Volume will not be marked Purged until the Volume retention period expires.
- Search the Pool for a Volume with VolStatus=Purged
- If a Pool named "Scratch" exists, search for a Volume 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 enabled If Python is enabled, a Python NewVolume event is generated before the Label Format directve is used. If the maximum number of Volumes specified for the pool is reached, a new Volume will not be created.

- Prune the oldest Volume if RecycleOldestVolume=yes (the Volume with the oldest LastWritten date and VolStatus equal to Full, Recycle, Purged, Used, or Append 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, Recycle, Purged, Used, or Append is chosen). We strongly recommend against the use of **PurgeOldestVolume** as it can quite easily lead to loss of current backup data.

- 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, it 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 used.

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 tapes 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 from Alan Brown 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 and the Volume retention period has expired, 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 tape with no files or jobs listed in the database, but which is still not marked as "purged".

22.4 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 created by Arno Lehmann:

```
A typical volume life cycle is like this:

              because job count or size limit exceeded
      Append  ----------------------------------------> Used
        ^                                                  |
        | First Job writes to       Retention time passed |
        | the volume                  and recycling takes |
        |                                           place |
        |                                                  v
      Recycled <------------------------------------- Purged
                     Volume is selected for reuse
```

22.5 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
}
```

22.6 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.

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 = XXXXXXXXXXXXX
  Device = FileStorage
  Media Type = 8mm
}
FileSet {
  Name = "File Set"
  Include = signature=MD5 {
    fffffffffffffffff
  }
  Exclude = { *.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
}
```

22.7 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 = XXXXXXXXXXX
  SDPort = 9103
  Password = XXXXXXXXXXXXX
  Device = FileStorage
  Media Type = File
}
FileSet {
  Name = "Test Files"
  Include = signature=MD5 {
    fffffffffffffffff
  }
  Exclude = { *.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 **ffffffffff**'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.

22.8 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 tape 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**.
- In Bacula version 1.30 or greater, 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 23

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.

23.1 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.

23.1.1 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, with all versions up to and including 1.39.28, you should probably limit the Volume size to some reasonable value such as say 5GB. This is because during a restore, Bacula is currently unable to seek to the proper place in a disk volume to restore a file, which means that it must read all records up to where the restore begins. If your Volumes are 50GB, reading half or more of the volume could take quite a bit of time. Also, if you ever have a partial hard disk failure, you are more likely to be able to recover more data if they are in smaller Volumes.

- 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.

23.1.2 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. In version 1.37 and later, it is probably much better to use Python scripting and the NewVolume event since generating Volume labels in a Python script is much easier than trying to figure out Counter variables. See the Python Scripting chapter of this manual for more details.

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 of this manual.

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.

23.1.3 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 records (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 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.

23.2 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.

23.3 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 = compression=GZIP signature=SHA1 {
    /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).

23.4 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 **/disk2** 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.

23.5 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 = compression=GZIP signature=SHA1 {
    /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 24

DVD Volumes

Bacula allows you to specify that you want to write to DVD. However, this feature is implemented only in version 1.37 or later. You may in fact write to DVD+RW, DVD+R, DVD-R, or DVD-RW media. The actual process used by Bacula is to first write the image to a spool directory, then when the Volume reaches a certain size or, at your option, at the end of a Job, Bacula will transfer the image from the spool directory to the DVD. The actual work of transferring the image is done by a script **dvd-handler**, and the heart of that script is a program called **growisofs** which allows creating or adding to a DVD ISO filesystem.

You must have **dvd+rw-tools** loaded on your system for DVD writing to work. Please note that the original **dvd+rw-tools** package does **NOT** work with Bacula. You must apply a patch which can be found in the **patches** directory of Bacula sources with the name **dvd+rw-tools-5.21.4.10.8.bacula.patch** for version 5.21 of the tools, or patch bf dvd+rw-tools-6.1.bacula.patch if you have version 6.1 on your system. Unfortunately, this requires you to build the dvd_rw-tools from source.

Note, some Linux distros such as Debian dvd+rw-tools-7.0-4 package already have the patch applied, so please check.

The fact that Bacula cannot use the OS to write directly to the DVD makes the whole process a bit more error prone than writing to a disk or a tape, but nevertheless, it does work if you use some care to set it up properly. However, at the current time (version 1.39.30 – 12 December 2006) we still consider this code to be BETA quality. As a consequence, please do careful testing before relying on DVD backups in production.

The remainder of this chapter explains the various directives that you can use to control the DVD writing.

24.1 DVD Specific SD Directives

The following directives are added to the Storage daemon's Device resource.

Requires Mount = *Yes—No* You must set this directive to **yes** for DVD-writers, and to **no** for all other devices (tapes/files). This directive indicates if the device requires to be mounted using the **Mount Command**. To be able to write a DVD, the following directives must also be defined: **Mount Point**, **Mount Command**, **Unmount Command** and **Write Part Command**.

Mount Point = *directory* Directory where the device can be mounted.

Mount Command = *name-string* Command that must be executed to mount the device. Although the device is written directly, the mount command is necessary in order to determine the free space left on the DVD. 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"
```

However, if you have defined a mount point in /etc/fstab, you might be able to use a mount command such as:

```
Mount Command = "/bin/mount /media/dvd"
```

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"
```

Write Part Command = *name-string* Command that must be executed to write a part to the device. Before the command is executed, %a is replaced with the Archive Device, %m with the Mount Point, %e is replaced with 1 if we are writing the first part, and with 0 otherwise, and %v with the current part filename.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Write Part Command = "/path/dvd-handler %a write %e %v"
```

Where **/path** is the path to your scripts install directory, and dvd-handler is the Bacula supplied script file. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

Free Space Command = *name-string* Command that must be executed to check how much free space is left on the device. Before the command is executed,%a is replaced with the Archive Device.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Free Space Command = "/path/dvd-handler %a free"
```

Where **/path** is the path to your scripts install directory, and dvd-freespace is the Bacula supplied script file. If you want to specify your own command, please look at the code in dvd-handler to see what output Bacula expects from this command. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

If you do not set it, Bacula will expect there is always free space on the device.

In addition to the directives specified above, you must also specify the other standard Device resource directives. Please see the sample DVD Device resource in the default bacula-sd.conf file. Be sure to specify the raw device name for **Archive Device**. It should be a name such as **/dev/cdrom** or **/media/cdrecorder** or **/dev/dvd** depending on your system. It will not be a name such as **/mnt/cdrom**.

Finally, for **growisofs** to work, it must be able to lock a certain amount of memory in RAM. If you have restrictions on this function, you may have failures. Under **bash**, you can set this with the following command:

```
ulimit -l unlimited
```

24.2 Edit Codes for DVD 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)
```

24.3 DVD Specific Director Directives

The following directives are added to the Director's Job resource.

Write Part After Job = <yes—no> If this directive is set to **yes** (default **no**), the Volume written to a temporary spool file for the current Job will be written to the DVD as a new part file will be created after the job is finished.

It should be set to **yes** when writing to devices that require a mount (for example DVD), so you are sure that the current part, containing this job's data, is written to the device, and that no data is left in the temporary file on the hard disk. However, on some media, like DVD+R and DVD-R, a lot of space (about 10Mb) is lost everytime a part is written. So, if you run several jobs each after another, you could set this directive to **no** for all jobs, except the last one, to avoid wasting too much space, but to ensure that the data is written to the medium when all jobs are finished.

This directive is ignored for devices other than DVDs.

24.4 Other Points

- Please be sure that you have any automatic DVD mounting disabled before running Bacula – this includes auto mounting in /etc/fstab, hotplug, ... If the DVD is automatically mounted by the OS, it will cause problems when Bacula tries to mount/unmount the DVD.
- Please be sure that you the directive **Write Part After Job** set to **yes**, otherwise the last part of the data to be written will be left in the DVD spool file and not written to the DVD. The DVD will then be unreadable until this last part is written. If you have a series of jobs that are run one at a time, you can turn this off until the last job is run.

- The current code is not designed to have multiple simultaneous jobs writing to the DVD. As a consequence, please ensure that only one DVD backup job runs at any time.

- Writing and reading of DVD+RW seems to work quite reliably provided you are using the patched dvd+rw-mediainfo programs. On the other hand, we do not have enough information to ensure that DVD-RW or other forms of DVDs work correctly.

- DVD+RW supports only about 1000 overwrites. Every time you mount the filesystem read/write will count as one write. This can add up quickly, so it is best to mount your DVD+RW filesystem read-only. Bacula does not need the DVD to be mounted read-write, since it uses the raw device for writing.

- Reformatting DVD+RW 10-20 times can apparently make the medium unusable. Normally you should not have to format or reformat DVD+RW media. If it is necessary, current versions of growisofs will do so automatically.

- We have had several problems writing to DVD-RWs (this does NOT concern DVD+RW), because these media have two writing-modes: **Incremental Sequential** and **Restricted Overwrite**. Depending on your device and the media you use, one of these modes may not work correctly (e.g. **Incremental Sequential** does not work with my NEC DVD-writer and Verbatim DVD-RW).

  To retrieve the current mode of a DVD-RW, run:

  ```
  dvd+rw-mediainfo /dev/xxx
  ```

  where you replace xxx with your DVD device name.

  Mounted Media line should give you the information.

  To set the device to **Restricted Overwrite** mode, run:

  ```
  dvd+rw-format /dev/xxx
  ```

  If you want to set it back to the default **Incremental Sequential** mode, run:

  ```
  dvd+rw-format -blank /dev/xxx
  ```

- Bacula only accepts to write to blank DVDs. To quickly blank a DVD+/-RW, run this command:

  ```
  dd if=/dev/zero bs=1024 count=512 | growisofs -Z /dev/xxx=/dev/fd/0
  ```

Then, try to mount the device, if it cannot be mounted, it will be considered as blank by Bacula, if it can be mounted, try a full blank (see below).

- If you wish to blank completely a DVD+/-RW, use the following:

  ```
  growisofs -Z /dev/xxx=/dev/zero
  ```

  where you replace xxx with your DVD device name. However, note that this blanks the whole DVD, which takes quite a long time (16 minutes on mine).

- DVD+RW and DVD-RW support only about 1000 overwrites (i.e. don't use the same medium for years if you don't want to have problems...).

  To write to the DVD the first time use:

  ```
  growisofs -Z /dev/xxx filename
  ```

  To add additional files (more parts use):

  ```
  growisofs -M /dev/xxx filename
  ```

  The option **-use-the-force-luke=4gms** was added in growisofs 5.20 to override growisofs' behavior of always checking for the 4GB limit. Normally, this option is recommended for all Linux 2.6.8 kernels or greater, since these newer kernels can handle writing more than 4GB. See below for more details on this subject.

- For more information about DVD writing, please look at the dvd+rw-tools homepage.

- According to bug #912, bscan cannot read multi-volume DVDs. This is on our TODO list, but unless someone submits a patch it is not likely to be done any time in the near future. (9 Sept 2007).

Chapter 25

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.

25.1 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.

25.2 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.

25.3 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.

25.3.1 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.

25.3.2 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.

25.3.3 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.

25.4 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 = "/home/bacula/bin/query.sql"
  WorkingDirectory = "/home/bacula/working"
  PidDirectory = "/home/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 = "/home/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 for details.
  RunBeforeJob = "/home/bacula/bin/make_catalog_backup bacula bacula"
  # This deletes the copy of the catalog
  RunAfterJob  = "/home/bacula/bin/delete_catalog_backup"
  Write Bootstrap = "/home/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 = /home/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 = "/home/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 = "/home/bacula/working"
  Pid Directory = "/home/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 26

Migration

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 section 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).

26.1 Migration 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**, in which case no Pool is used, unless you specifically include it in the SQL query. Note, the Pool resource referenced 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.

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.

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 approximative 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.

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.

26.2 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.

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, ...).

26.3 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.

- Jobs on Volumes will be 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.

- 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.

- 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 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.

- 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.

- Migration is only implemented for a single Storage daemon. You cannot read on one Storage daemon and write on another.

26.4 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 = "ccV3lVTsQRsdIUGyab0N4sMDavui2hOBkmpBU0aQKOr9"
  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.

Chapter 27

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.

27.1 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.

27.1.1 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.

27.1.2 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.

27.1.3 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.

27.2 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
```

```
/full-path/bconsole -c /full-path/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
/full-path/bconsole -c /full-path/bconsole.conf <\&ltEND_OF_DATA
unmount storage=your-storage-name
END_OF_DATA
# the following is a shell command
mt eject
/full-path/bconsole -c /full-path/bconsole.conf <<END_OF_DATA
mount storage=your-storage-name
END_OF_DATA
```

27.3 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.

27.3.1 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.

27.3.2 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.

27.3.3 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 = "/var/lib/bacula"
  PidDirectory = "/var/run"
  SubSysDirectory = "/var/lock/subsys"
  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 = "/var/lib/bacula/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 for details.
  RunBeforeJob = "/usr/lib/bacula/make_catalog_backup -u bacula"
  # This deletes the copy of the catalog, and ejects the tape
  RunAfterJob  = "/etc/bacula/end_of_backup.sh"
  Write Bootstrap = "/var/lib/bacula/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 = signature=MD5 {
    /
    /data
  }
  Exclude = { /proc /tmp /.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 = signature=MD5 {
     /var/lib/bacula/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 = "/usr/sbin/bsmtp -h localhost -f \"\(Bacula\) %r\"
     -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/usr/sbin/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 = "/var/lib/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
/usr/lib/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 28

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. You can find a list of autochangers supported by **mtx** at the following link: http://mtx.opensource-sw.net/compatibility.php. The home page for the **mtx** project can be found at: http://mtx.opensource-sw.net/.

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.

28.1 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.

For more detailed information on what SCSI devices you have please see the Linux SCSI Tricks section of the Tape Testing chapter of this manual.

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.

28.2 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.

28.3 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.

28.4 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, 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.

28.5 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 = "/etc/bacula/mtx-changer %c %o %S %a %d"
```

Where you will need to adapt the **/etc/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

```
Device 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 29

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 = "/etc/bacula/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.

29.1 An Example Configuration File

The following two resources implement an autochanger:

```
Autochanger {
  Name = "Autochanger"
  Device = DDS-4
  Changer Device = /dev/sg0
  Changer Command = "/etc/bacula/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.

29.2 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 = "/etc/bacula/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.

29.3 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:

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).

```
Pool {
  Name ...
  Cleaning Prefix = "CLN"
}
```

Any slot containing a barcode of CLNxxxx will be treated as a cleaning tape and will not be mounted.

29.4 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.

29.5 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.

29.6 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 *\(
```

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.

29.7 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.

29.8 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 of this manual for **important** information concerning your tape drive before doing the autochanger testing.

29.9 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 **/etc/bacula/mtx-changer**):

Make sure Bacula is not running.

/etc/bacula/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**.

/etc/bacula/mtx-changer /dev/sg0 slots This command should return the number of slots in your autochanger.

/etc/bacula/mtx-changer /dev/sg0 unload 1 /dev/nst0 0 If a tape is loaded from slot 1, this should cause it to be unloaded.

/etc/bacula/mtx-changer /dev/sg0 load 3 /dev/nst0 0 Assuming you have a tape in slot 3, it will be loaded into drive (0).

/etc/bacula/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.

/etc/bacula/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
/etc/bacula/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
/etc/bacula/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
/etc/bacula/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
mt -f /dev/st0 offline
/etc/bacula/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.

29.10 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:8101
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:

```
*{\bf 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    |
| ...                                                                            |
+-------+----------+--------+---------+-------+--------+----------+-------+------+
```

29.11 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.

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.

29.12 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 {\bf slot} is the non-zero integer representing
              the slot number, and {\bf 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 30

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. You can find a list of autochangers supported by **mtx** at the following link: http://mtx.opensource-sw.net/compatibility.php. The home page for the **mtx** project can be found at: http://mtx.opensource-sw.net/.

OS	Man.	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	??	Tape	??	??
Linux	Dell	DLT VI,LTO-2,LTO3	PowerVault 122T/132T/136T	-	100GB
Linux	Dell	LTO-2	PowerVault 124T	-	200GB
-	DFSMS	??	VM RMM	-	??
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
Linux	HP (Compaq)	DLT VI	Compaq TL-895	96+4 import export	35/70GB
z/VM	IBM	??	IBM Tape Manager	-	??
z/VM	IBM	??	native tape	-	??
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	???	IBM TotalStorage 3582L23	??	??

Debian	Overland	LTO	Overland Load-erXpress LTO/DLT8000	10-19	40-100GB
Fedora	Overland	LTO	Overland Power-Loader LTO-2	10-19	200/400GB
FreeBSD 5.4-Stable	Overland	LTO-2	Overland Power-loader 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	??	??
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
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	??	ACSLS	??	??
Solaris	Sun	4mm DLT	Sun Desk-top Archive Python 29279	4	20GB

Linux	Tandberg	DLT VI	VS 640	8?	35/70GB
Linux 2.6.x	Tandberg Data	SLR100	SLR100 Autoloader	8	50/100GB

Chapter 31

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.

31.1 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 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 conf 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 conf file (default, the working directory).

  Spool Directory = directory

31.2 !!! MAJOR WARNING !!!

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.

31.3 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 32

Python Scripting

You may be asking what Python is and why a scripting language is needed in Bacula. The answer to the first question is that Python is an Object Oriented scripting language with features similar to those found in Perl, but the syntax of the language is much cleaner and simpler. The answer to why have scripting in Bacula is to give the user more control over the whole backup process. Probably the simplest example is when Bacula needs a new Volume name, with a scripting language such as Python, you can generate any name you want, based on the current state of Bacula.

32.1 Python Configuration

Python must be enabled during the configuration process by adding a --with-python, and possibly specifying an alternate directory if your Python is not installed in a standard system location. If you are using RPMs you will need the python-devel package installed.

When Python is configured, it becomes an integral part of Bacula and runs in Bacula's address space, so even though it is an interpreted language, it is very efficient.

When the Director starts, it looks to see if you have a **Scripts Directory** Directive defined (normal default **/etc/bacula/scripts**, if so, it looks in that directory for a file named **DirStartUp.py**. If it is found, Bacula will pass this file to Python for execution. The **Scripts Directory** is a new directive that you add to the Director resource of your bacula-dir.conf file.

Note: Bacula does not install Python scripts by default because these scripts are for you to program. This means that with a default installation with Python enabled, Bacula will print the following error message:

```
09-Jun 15:14 bacula-dir: ERROR in pythonlib.c:131 Could not import
Python script /etc/bacula/scripts/DirStartUp. Python disabled.
```

The source code directory **examples/python** contains sample scripts for DirStartUp.py, SDStartUp.py, and FDStartUp.py that you might want to use as a starting point. Normally, your scripts directory (at least where you store the Python scripts) should be writable by Bacula, because Python will attempt to write a compiled version of the scripts (e.g. DirStartUp.pyc) back to that directory.

When starting with the sample scripts, you can delete any part that you will not need, but you should keep all the Bacula Event and Job Event definitions. If you do not want a particular event, simply replace the existing code with a **noop = 1**.

32.2 Bacula Events

A Bacula event is a point in the Bacula code where Bacula will call a subroutine (actually a method) that you have defined in the Python StartUp script. Events correspond to some significant event such as a Job Start, a Job End, Bacula needs a new Volume Name, ... When your script is called, it will have access to all the Bacula variables specific to the Job (attributes of the Job Object), and it can even call some of the Job methods (subroutines) or set new values in the Job attributes, such as the Priority. You will see below how the events are used.

32.3 Python Objects

There are four Python objects that you will need to work with:

The Bacula Object The Bacula object is created by the Bacula daemon (the Director in the present case) when the daemon starts. It is available to the Python startup script, **DirStartup.py**, by importing the Bacula definitions with **import bacula**. The methods available with this object are described below.

The Bacula Events Class You create this class in the startup script, and you pass it to the Bacula Object's **set_events** method. The purpose of the Bacula Events Class is to define what global or daemon events you want to monitor. When one of those events occurs, your Bacula Events Class will be called at the method corresponding to the event. There are currently three events, JobStart, JobEnd, and Exit, which are described in detail below.

The Job Object When a Job starts, and assuming you have defined a JobStart method in your Bacula Events Class, Bacula will create a Job Object. This object will be passed to the JobStart event. The Job Object has a has good number of read-only members or attributes providing many details of the Job, and it also has a number of writable attributes that allow you to pass information into the Job. These attributes are described below.

The Job Events Class You create this class in the JobStart method of your Bacula Events class, and it allows you to define which of the possible Job Object events you want to see. You must pass an instance of your Job Events class to the Job Object set_events() method. Normally, you will probably only have one Job Events Class, which will be instantiated for each Job. However, if you wish to see different events in different Jobs, you may have as many Job Events classes as you wish.

The first thing the startup script must do is to define what global Bacula events (daemon events), it wants to see. This is done by creating a Bacula Events class, instantiating it, then passing it to the **set_events** method. There are three possible events.

JobStart This Python method, if defined, will be called each time a Job is started. The method is passed the class instantiation object as the first argument, and the Bacula Job object as the second argument. The Bacula Job object has several built-in methods, and you can define which ones you want called. If you do not define this method, you will not be able to interact with Bacula jobs.

JobEnd This Python method, if defined, will be called each time a Job terminates. The method is passed the class instantiation object as the first argument, and the Bacula Job object as the second argument.

Exit This Python method, if defined, will be called when the Director terminates. The method is passed the class instantiation object as the first argument.

Access to the Bacula variables and methods is done with:

import bacula

The following are the read-only attributes provided by the bacula object.

Name

ConfigFile

WorkingDir

Version string consisting of "Version Build-date"

A simple definition of the Bacula Events Class might be the following:

```
import sys, bacula
class BaculaEvents:
  def JobStart(self, job):
     ...
```

Then to instantiate the class and pass it to Bacula, you would do:

```
bacula.set_events(BaculaEvents()) # register Bacula Events wanted
```

And at that point, each time a Job is started, your BaculaEvents JobStart method will be called.

Now to actually do anything with a Job, you must define which Job events you want to see, and this is done by defining a JobEvents class containing the methods you want called. Each method name corresponds to one of the Job Events that Bacula will generate.

A simple Job Events class might look like the following:

```
class JobEvents:
  def NewVolume(self, job):
     ...
```

Here, your JobEvents class method NewVolume will be called each time the Job needs a new Volume name. To actually register the events defined in your class with the Job, you must instantiate the JobEvents class and set it in the Job **set_events** variable. Note, this is a bit different from how you registered the Bacula events. The registration process must be done in the Bacula JobStart event (your method). So, you would modify Bacula Events (not the Job events) as follows:

```
import sys, bacula
class BaculaEvents:
  def JobStart(self, job):
     events = JobEvents()         # create instance of Job class
     job.set_events(events)       # register Job events desired
     ...
```

When a job event is triggered, the appropriate event definition is called in the JobEvents class. This is the means by which your Python script or code gets control. Once it has control, it may read job attributes, or set them. See below for a list of read-only attributes, and those that are writable.

In addition, the Bacula **job** object in the Director has a number of methods (subroutines) that can be called. They are:

set_events The set_events method takes a single argument, which is the instantiation of the Job Events class that contains the methods that you want called. The method names that will be called must correspond to the Bacula defined events. You may define additional methods but Bacula will not use them.

run The run method takes a single string argument, which is the run command (same as in the Console) that you want to submit to start a new Job. The value returned by the run method is the JobId of the job that started, or -1 if there was an error.

write The write method is used to be able to send print output to the Job Report. This will be described later.

cancel The cancel method takes a single integer argument, which is a JobId. If JobId is found, it will be canceled.

DoesVolumeExist The DoesVolumeExist method takes a single string argument, which is the Volume name, and returns 1 if the volume exists in the Catalog and 0 if the volume does not exist.

The following attributes are read/write within the Director for the **job** object.

Priority Read or set the Job priority. Note, that setting a Job Priority is effective only before the Job actually starts.

Level This attribute contains a string representing the Job level, e.g. Full, Differential, Incremental, ... if read. The level can also be set.

The following read-only attributes are available within the Director for the **job** object.

Type This attribute contains a string representing the Job type, e.g. Backup, Restore, Verify, ...

JobId This attribute contains an integer representing the JobId.

Client This attribute contains a string with the name of the Client for this job.

NumVols This attribute contains an integer with the number of Volumes in the Pool being used by the Job.

Pool This attribute contains a string with the name of the Pool being used by the Job.

Storage This attribute contains a string with the name of the Storage resource being used by the Job.

Catalog This attribute contains a string with the name of the Catalog resource being used by the Job.

MediaType This attribute contains a string with the name of the Media Type associated with the Storage resource being used by the Job.

Job This attribute contains a string containing the name of the Job resource used by this job (not unique).

JobName This attribute contains a string representing the full unique Job name.

JobStatus This attribute contains a single character string representing the current Job status. The status may change during execution of the job. It may take on the following values:

- **C** Created, not yet running
- **R** Running
- **B** Blocked
- **T** Completed successfully
- **E** Terminated with errors
- **e** Non-fatal error
- **f** Fatal error
- **D** Verify found differences
- **A** Canceled by user

- **F** Waiting for Client
- **S** Waiting for Storage daemon
- **m** Waiting for new media
- **M** Waiting for media mount
- **s** Waiting for storage resource
- **j** Waiting for job resource
- **c** Waiting for client resource
- **d** Waiting on maximum jobs
- **t** Waiting on start time
- **p** Waiting on higher priority jobs

Priority This attribute contains an integer with the priority assigned to the job.

CatalogRes tuple consisting of (DBName, Address, User, Password, Socket, Port, Database Vendor) taken from the Catalog resource for the Job with the exception of Database Vendor, which is one of the following: MySQL, PostgreSQL, SQLite, Internal, depending on what database you configured.

VolumeName After a Volume has been purged, this attribute will contain the name of that Volume. At other times, this value may have no meaning.

The following write-only attributes are available within the Director:

JobReport Send line to the Job Report.

VolumeName Set a new Volume name. Valid only during the NewVolume event.

32.4 Python Console Command

There is a new Console command named **python**. It takes a single argument **restart**. Example:

```
python restart
```

This command restarts the Python interpreter in the Director. This can be useful when you are modifying the DirStartUp script, because normally Python will cache it, and thus the script will be read one time.

32.5 Debugging Python Scripts

In general, you debug your Python scripts by using print statements. You can also develop your script or important parts of it as a separate file using the Python interpreter to run it. Once you have it working correctly, you can then call the script from within the Bacula Python script (DirStartUp.py).

If you are having problems loading DirStartUp.py, you will probably not get any error messages because Bacula can only print Python error messages after the Python interpreter is started. However, you may be able to see the error messages by starting Bacula in a shell window with the **-d1** option on the command line. That should cause the Python error messages to be printed in the shell window.

If you are getting error messages such as the following when loading DirStartUp.py:

```
Traceback (most recent call last):
  File "/etc/bacula/scripts/DirStartUp.py", line 6, in ?
    import time, sys, bacula
ImportError: /usr/lib/python2.3/lib-dynload/timemodule.so: undefined
symbol: PyInt_FromLong
bacula-dir: pythonlib.c:134 Python Import error.
```

It is because the DirStartUp script is calling a dynamically loaded module (timemodule.so in the above case) that then tries to use Python functions exported from the Python interpreter (in this case PyInt_FromLong). The way Bacula is currently linked with Python does not permit this. The solution to the problem is to put such functions (in this case the import of time into a separate Python script, which will do your calculations and return the values you want. Then call (not import) this script from the Bacula DirStartUp.py script, and it all should work as you expect.

32.6 Python Example

An example script for the Director startup file is provided in **examples/python/DirStartup.py** as follows:

```
#
# Bacula Python interface script for the Director
#
```

```
# You must import both sys and bacula
import sys, bacula

# This is the list of Bacula daemon events that you
#  can receive.
class BaculaEvents(object):
  def __init__(self):
     # Called here when a new Bacula Events class is
     #  is created. Normally not used
     noop = 1

  def JobStart(self, job):
     """
       Called here when a new job is started. If you want
       to do anything with the Job, you must register
       events you want to receive.
     """
     events = JobEvents()         # create instance of Job class
     events.job = job             # save Bacula's job pointer
     job.set_events(events)       # register events desired
     sys.stderr = events          # send error output to Bacula
     sys.stdout = events          # send stdout to Bacula
     jobid = job.JobId; client = job.Client
     numvols = job.NumVols
     job.JobReport="Python Dir JobStart: JobId=%d Client=%s NumVols=%d\n" % (jobid,client,numvols)

  # Bacula Job is going to terminate
  def JobEnd(self, job):
     jobid = job.JobId
     client = job.Client
     job.JobReport="Python Dir JobEnd output: JobId=%d Client=%s.\n" % (jobid, client)

  # Called here when the Bacula daemon is going to exit
  def Exit(self, job):
      print "Daemon exiting."

bacula.set_events(BaculaEvents()) # register daemon events desired

"""
  These are the Job events that you can receive.
"""
class JobEvents(object):
  def __init__(self):
     # Called here when you instantiate the Job. Not
     # normally used
     noop = 1

  def JobInit(self, job):
     # Called when the job is first scheduled
     noop = 1

  def JobRun(self, job):
     # Called just before running the job after initializing
```

```
      #  This is the point to change most Job parameters.
      #  It is equivalent to the JobRunBefore point.
      noop = 1

   def NewVolume(self, job):
      # Called when Bacula wants a new Volume name. The Volume
      #  name returned, if any, must be stored in job.VolumeName
      jobid = job.JobId
      client = job.Client
      numvol = job.NumVols;
      print job.CatalogRes
      job.JobReport = "JobId=%d Client=%s NumVols=%d" % (jobid, client, numvol)
      job.JobReport="Python before New Volume set for Job.\n"
      Vol = "TestA-%d" % numvol
      job.JobReport = "Exists=%d TestA-%d" % (job.DoesVolumeExist(Vol), numvol)
      job.VolumeName="TestA-%d" % numvol
      job.JobReport="Python after New Volume set for Job.\n"
      return 1

   def VolumePurged(self, job):
      # Called when a Volume is purged. The Volume name can be referenced
      #  with job.VolumeName
      noop = 1
```

Chapter 33

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.

33.1 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.

33.2 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 34

Bacula Frequently Asked Questions

These are questions that have been submitted over time by the Bacula users. The following FAQ is very useful, but it is not always up to date with newer information, so after reading it, if you don't find what you want, you might try the Bacula wiki maintained by Frank Sweetser, which contains more than just a FAQ: http://wiki.bacula.org or go directly to the FAQ at: http://wiki.bacula.org/doku.php?id=faq.

Please also see the bugs section of this document for a list of known bugs and solutions.

34.1 What is Bacula?

What is Bacula? **Bacula** is a network backup and restore program.

34.2 Does Bacula support Windows?

Does Bacula support Windows? Yes, Bacula compiles and runs on Windows machines (Win98, WinMe, WinXP, WinNT, Win2003, and Win2000). We provide a binary version of the Client (bacula-fd), but have not tested the Director nor the Storage daemon. Note, Win95 is no longer supported because it doesn't have the GetFileAttributesExA API call.

34.3 What language is Bacula written in?

What language is Bacula written in? It is written in C++, but it is mostly C code using only a limited set of the C++ extensions over C. Thus Bacula is completely compiled using the C++ compiler. There are several modules, including the Win32 interface, that are written using the object oriented C++ features. Over time, we are slowly adding a larger subset of C++.

34.4 On what machines does Bacula run?

On what machines does Bacula run? **Bacula** builds and executes on Red Hat Linux (versions RH7.1-RHEL 4.0, Fedora, SuSE, Gentoo, Debian, Mandriva, ...), FreeBSD, Solaris, Alpha, SGI (client), NetBSD, OpenBSD, Mac OS X (client), and Win32.

Bacula has been my only backup tool for over seven years backing up 8 machines nightly (6 Linux boxes running SuSE, previously Red Hat and Fedora, a WinXP machine, and a WinNT machine).

34.5 Is Bacula Stable?

Is Bacula Stable? Yes, it is remarkably stable, but remember, there are still a lot of unimplemented or partially implemented features. With a program of this size (150,000+ lines of C++ code not including the SQL programs) there are bound to be bugs. The current test environment (a twisted pair local network and a HP DLT backup tape) is not exactly ideal, so additional testing on other sites is necessary. The File daemon has never crashed – running months at a time with no intervention. The Storage daemon is remarkably stable with most of the problems arising during labeling or switching tapes. Storage daemon crashes are rare but running multiple drives and simultaneous jobs sometimes (rarely) problems. The Director, given the multitude of functions it fulfills is also relatively stable. In a production environment, it rarely if ever crashes. Of the three daemons, the Director is the most prone to having problems. Still, it frequently runs several months with no problems.

There are a number of reasons for this stability.

1. The program is constantly checking the chain of allocated memory buffers to ensure that no overruns have occurred.

2. All memory leaks (orphaned buffers) are reported each time the program terminates.

3. Any signal (segmentation fault, ...) generates a traceback that is emailed to the developer. This permits quick resolution of bugs even if they only show up rarely in a production system.

4. There is a reasonably comprehensive set of regression tests that avoids re-creating the most common errors in new versions of Bacula.

34.6 I'm Getting Authorization Errors. What is Going On?

I'm Getting Authorization Errors. What is Going On? For security reasons, Bacula requires that both the File daemon and the Storage daemon know the name of the Director as well as its password. As a consequence, if you change the Director's name or password, you must make the corresponding change in the Storage daemon's and in the File daemon's configuration files.

During the authorization process, the Storage daemon and File daemon also require that the Director authenticates itself, so both ends require the other to have the correct name and password.

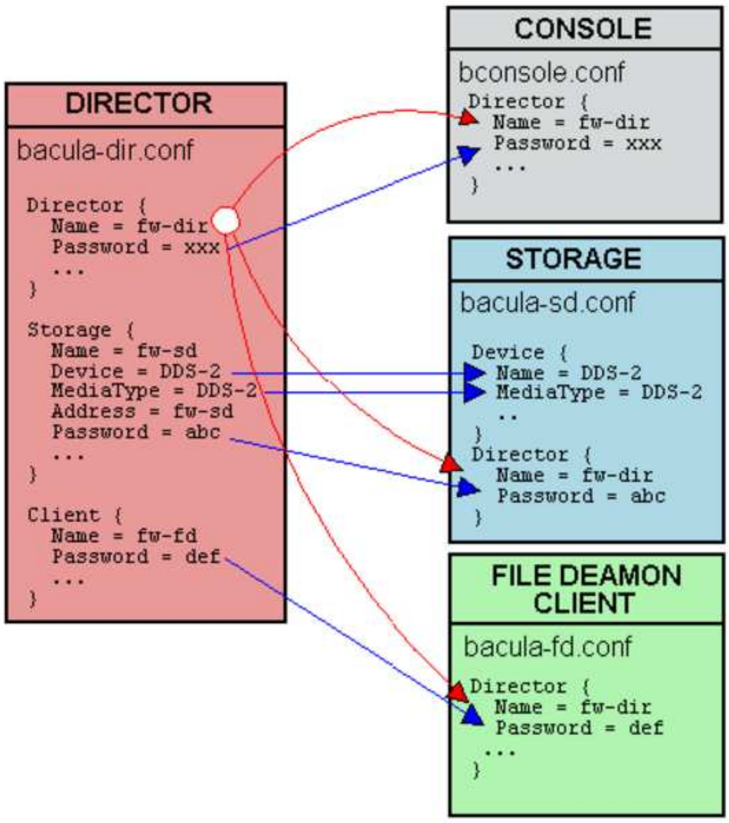
If you have edited the conf files and modified any name or any password, and you are getting authentication errors, then your best bet is to go back to the original conf files generated by the Bacula installation process. Make only the absolutely necessary modifications to these files – e.g. add the correct email address. Then follow the instructions in the Running Bacula chapter of this manual. You will run a backup to disk and a restore. Only when that works, should you begin customization of the conf files.

Another reason that you can get authentication errors is if you are running Multiple Concurrent Jobs in the Director, but you have not set them in the File daemon or the Storage daemon. Once you reach their limit, they will reject the connection producing authentication (or connection) errors.

If you are having problems connecting to a Windows machine that previously worked, you might try restarting the Bacula service since Windows frequently encounters networking connection problems.

Some users report that authentication fails if there is not a proper reverse DNS lookup entry for the machine. This seems to be a requirement of gethostbyname(), which is what Bacula uses to translate names into IP addresses. If you cannot add a reverse DNS entry, or you don't know how to do so, you can avoid the problem by specifying an IP address rather than a machine name in the appropriate Bacula conf file.

Here is a picture that indicates what names/passwords in which files/Resources must match up:



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**. The right column is where the corresponding values should be found in the Console, Storage daemon (SD), and File daemon (FD) configuration files.

Another thing to check is to ensure that the Bacula component you are trying to access has **Maximum Concurrent Jobs** set large enough to handle each of the Jobs and the Console that want to connect simultaneously. Once the maximum connections has been reached, each Bacula component will reject all new connections.

Finally, make sure you have no **hosts.allow** or **hosts.deny** file that is not permitting access to the site trying to connect.

34.7 Bacula Runs Fine but Cannot Access a Client on a Different Machine. Why?

Bacula Runs Fine but Cannot Access a Client on a Different Machine. Why? There are several reasons why Bacula could not contact a client on a different machine. They are:

- It is a Windows Client, and the client died because of an improper configuration file. Check that the Bacula icon is in the system tray and the the menu items work. If the client has died, the icon will disappear only when you move the mouse over the icon.
- The Client address or port is incorrect or not resolved by DNS. See if you can ping the client machine using the same address as in the Client record.
- You have a firewall, and it is blocking traffic on port 9102 between the Director's machine and the Client's machine (or on port 9103 between the Client and the Storage daemon machines).
- Your password or names are not correct in both the Director and the Client machine. Try configuring everything identical to how you run the client on the same machine as the Director, but just change the Address. If that works, make the other changes one step at a time until it works.
- You may also be having problems between your File daemon and your Storage daemon. The name you use in the Storage resource of your Director's conf file must be known (resolvable) by the File daemon, because it is passed symbolically to the File daemon, which then resolves it to get an IP address used to contact the Storage daemon.
- You may have a **hosts.allow** or **hosts.deny** file that is not permitting access.

34.8 My Catalog is Full of Test Runs, How Can I Start Over?

My Catalog is Full of Test Runs, How Can I Start Over? If you are using MySQL do the following:

```
cd <bacula-source>/src/cats
./drop_mysql_tables
./make_mysql_tables
```

If you are using SQLite, do the following:

```
Delete bacula.db from your working directory.
cd <bacula-source>/src/cats
./drop_sqlite_tables
./make_sqlite_tables
```

Then write an EOF on each tape you used with **Bacula** using:

```
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

where you need to adjust the device name for your system.

34.9 I Run a Restore Job and Bacula Hangs. What do I do?

I Run a Restore Job and Bacula Hangs. What do I do? On Bacula version 1.25 and prior, it expects you to have the correct tape mounted prior to a restore. On Bacula version 1.26 and higher, it will ask you for the tape, and if the wrong one is mounted, it will inform you.

If you have previously done an **unmount** command, all Storage daemon sessions (jobs) will be completely blocked from using the drive unmounted, so be sure to do a **mount** after your unmount. If in doubt, do a second **mount**, it won't cause any harm.

34.10 I Cannot Get My Windows Client to Start Automatically?

I Cannot Get My Windows Client to Start Automatically? You are probably having one of two problems: either the Client is dying due to an incorrect configuration file, or you didn't do the Installation commands necessary to install it as a Windows Service.

For the first problem, see the next FAQ question. For the second problem, please review the Windows Installation instructions in this manual.

34.11 My Windows Client Immediately Dies When I Start It

My Windows Client Immediately Dies When I Start It The most common problem is either that the configuration file is not where it expects it to be, or that there is an error in the configuration file. You must have the configuration file in **c:\bacula\bin\bacula-fd.conf**.

To **see** what is going on when the File daemon starts on Windows, do the following:

```
Start a DOS shell Window.
cd c:\bacula\bin
bacula-fd -d100 -c c:\bacula\bin\bacula-fd.conf
```

This will cause the FD to write a file **bacula.trace** in the current directory, which you can examine and thereby determine the problem.

When I Start the Console, the Error Messages Fly By. How can I see them? Either use a shell window with a scroll bar, or use the gnome-console. In any case, you probably should be logging all output to a file, and then you can simply view the file using an editor or the **less** program. To log all output, I have the following in my Director's Message resource definition:

```
append = "/home/kern/bacula/bin/log" = all, !skipped
```

Obviously you will want to change the filename to be appropriate for your system.

34.12 My backups are not working on my Windows Client. What should I do?

I didn't realize that the backups were not working on my Windows Client. What should I You should be sending yourself an email message for each job. This will avoid the possibility of not knowing about a failed backup. To do so put something like:

```
Mail = yourname@yourdomain = all, !skipped
```

in your Director's message resource. You should then receive one email for each Job that ran. When you are comfortable with what is going on (it took me 9 months), you might change that to:

```
MailOnError = yourname@yourdomain = all, !skipped
```

then you only get email messages when a Job errors as is the case for your Windows machine.

You should also be logging the Director's messages, please see the previous FAQ for how to do so.

34.13 All my Jobs are scheduled for the same time. Will this cause problems?

All my Jobs are scheduled for the same time. Will this cause problems? No, not at all. Bacula will schedule all the Jobs at the same time, but will run them one after another unless you have increased the number of simultaneous jobs in the configuration files for the Director, the File daemon, and the Storage daemon. The appropriate configuration record is **Maximum Concurrent Jobs = nn**. At the current time, we recommend that you leave this set to **1** for the Director.

34.14 Can Bacula Backup My System To Files instead of Tape?

Can Bacula Backup My System To Files instead of Tape? Yes, in principle, Bacula can backup to any storage medium as long as you have correctly defined that medium in the Storage daemon's Device resource. For an example of how to backup to files, please see the Pruning Example in the Recycling chapter of this manual. Also, there is a whole chapter devoted to Basic Volume Management. This chapter was originally written to explain how to write to disk, but was expanded to include volume management. It is, however, still quite a good chapter to read.

34.15 Can I use a dummy device to test the backup?

Yes, to have a *Virtual* device which just consumes data, you can use a FIFO device (see Stored configuration). It's useful to test a backup.

```
Device {
  Name = NULL
  Media Type = NULL
  Device Type = Fifo
  Archive Device = /dev/null
  LabelMedia = yes
  Random Access = no
  AutomaticMount = no
  RemovableMedia = no
  MaximumOpenWait = 60
  AlwaysOpen = no
}
```

34.16 Can Bacula Backup and Restore Files Bigger than 2 Gigabytes?

Can Bacula Backup and Restore Files Bigger than 2 Gigabytes?
If your operating system permits it, and you are running Bacula version 1.26 or later, the answer is yes. To the best of our knowledge all client system supported by Bacula can handle files bigger 2 Gigabytes.

34.17 I want to stop a job.

I Started A Job then Decided I Really Did Not Want to Run It. Is there a better way tha
Yes, you normally should use the Console command **cancel** to cancel a Job that is either scheduled or running. If the Job is scheduled, it will be marked for cancellation and will be canceled when it is scheduled to start. If it is running, it will normally terminate after a few minutes. If the Job is waiting on a tape mount, you may need to do a **mount** command before it will be canceled.

34.18 Why have You Trademarked the Name Bacula?

Why have You Trademarked the Name Bacula®? We have trademarked the name Bacula to ensure that all media written by any program named Bacula will always be compatible. Anyone may use the name Bacula, even in a derivative product as long as it remains totally compatible in all respects with the program defined here.

34.19 Why is the Online Document for Version 1.39 but the Released Version is 1.38?

Why is the Online Document for Version 1.39 of Bacula when the Current Version

As Bacula is being developed, the document is also being enhanced, more often than not it has clarifications of existing features that can be very useful to our users, so we publish the very latest document. Fortunately it is rare that there are confusions with new features.

If you want to read a document that pertains only to a specific version, please use the one distributed in the source code. The web site also has online versions of both the released manual and the current development manual.

34.20 Does Bacula really save and restore all files?

How Can I Be Sure that Bacula Really Saves and Restores All Files?

It is really quite simple, but took me a while to figure out how to "prove" it. First make a Bacula Rescue disk, see the Disaster Recovery Using Bacula chapter of this manual. Second, you run a full backup of all your files on all partitions. Third, you run an Verify InitCatalog Job on the same FileSet, which effectively makes a record of all the files on your system. Fourth, you run a Verify Catalog job and assure yourself that nothing has changed (well, between an InitCatalog and Catalog one doesn't expect anything). Then do the unthinkable, write zeros on your MBR (master boot record) wiping out your hard disk. Now, restore your whole system using your Bacula Rescue disk and the Full backup you made, and finally re-run the Verify Catalog job. You will see that with the exception of the directory modification and access dates and the files changed during the boot, your system is identical to what it was before you wiped your hard disk. Alternatively you could do the wiping and restoring to another computer of the same type.

34.21 I want an Incremental but Bacula runs it as a Full backup. Why?

I did a Full backup last week, but now in running an Incremental, Bacula says it d

Before doing an Incremental or a Differential backup, Bacula checks to see if there was a prior Full backup of the same Job that terminated successfully. If so, it uses the date that full backup started as the time for comparing if files have changed. If Bacula does not find a successful full backup, it proceeds to do one. Perhaps you canceled the full backup, or it terminated in error. In such cases, the full backup will not be successful. You can check by entering **list jobs** and look to see if there is a prior Job with the same Name that has Level F and JobStatus T (normal termination).

Another reason why Bacula may not find a suitable Full backup is that every time you change the FileSet, Bacula will require a new Full backup. This is necessary to ensure that all files are properly backed up in the case where you have added more files to the FileSet. Beginning with version 1.31, the FileSets are also dated when they are created, and this date is displayed with the name when you are listing or selecting a FileSet. For more on backup levels see below.

See also **Ignore FileSet Changes** in the FileSet Resource definition in the Director chapter of this document.

34.22 Do you really handle unlimited path lengths?

How Can You Claim to Handle Unlimited Path and Filename Lengths when All Other Pro
Most of those other programs have been around for a long time, in fact since the beginning of Unix, which means that they were designed for rather small fixed length path and filename lengths. Over the years, these restrictions have been relaxed allowing longer names. Bacula on the other hand was designed in 2000, and so from the start, Path and Filenames have been kept in buffers that start at 256 bytes in length, but can grow as needed to handle any length. Most of the work is carried out by lower level routines making the coding rather easy.

Note that due to limitations Win32 path and filenames cannot exceed 260 characters. By using Win32 Unicode functions, we will remove this restriction in later versions of Bacula.

34.23 What Is the Really Unique Feature of Bacula?

What Is the Really Unique Feature of Bacula? Well, it is hard to come up with unique features when backup programs for Unix machines have been around since the 1960s. That said, I believe that

Bacula is the first and only program to use a standard SQL interface to catalog its database. Although this adds a bit of complexity and possibly overhead, it provides an amazingly rich set of features that are easy to program and enhance. The current code has barely scratched the surface in this regard (version 1.38).

The second feature, which gives a lot of power and flexibility to Bacula is the Bootstrap record definition.

The third unique feature, which is currently (1.30) unimplemented, and thus can be called vaporware :-), is Base level saves. When implemented, this will enormously reduce tape usage.

34.24 How can I force one job to run after another?

If I Run Multiple Simultaneous Jobs, How Can I Force One Particular Job to Run

Yes, you can set Priorities on your jobs so that they run in the order you specify. Please see: the Priority record in the Job resource.

34.25 I Am Not Getting Email Notification, What Can I Do?

I Am Not Getting Email Notification, What Can I Do? The most common problem is that you have not specified a fully qualified email address and your bsmtp server is rejecting the mail. The next most common problem is that your bsmtp server doesn't like the syntax on the From part of the message. For more details on this and other problems, please see the Getting Email Notification to Work section of the Tips chapter of this manual. The section Getting Notified of Job Completion of the Tips chapter may also be useful. For more information on the **bsmtp** mail program, please see bsmtp in the Volume Utility Tools chapter of this manual.

34.26 My retention periods don't work

I Change Recycling, Retention Periods, or File Sizes in my Pool Resource and the

The different variables associated with a Pool are defined in the Pool Resource, but are actually read by Bacula from the Catalog database. On Bacula versions prior to 1.30, after changing your Pool Resource, you must manually update the corresponding values in the Catalog by using the **update pool** command in the Console program. In Bacula version 1.30, Bacula does this for you automatically every time it starts.

When Bacula creates a Media record (Volume), it uses many default values from the Pool record. If you subsequently change the Pool record, the new values will be used as a default for the next Volume that is created, but if you want the new values to apply to existing Volumes, you must manually update the Volume Catalog entry using the **update volume** command in the Console program.

34.27 Why aren't my files compressed?

I Have Configured Compression On, But None of My Files Are Compressed. Why?
There are two kinds of compression. One is tape compression. This is done by the tape drive hardware, and you either enable or disable it with system tools such as **mt**. This compression works independently of Bacula, and when it is enabled, you should not use the Bacula software compression.

Bacula also has software compression code in the File daemons, which you normally need to enable only when backing up to file Volumes. There are two conditions necessary to enable the Bacula software compression.

1. You must have the zip development libraries loaded on your system when building Bacula and Bacula must find this library, normally **/usr/lib/libz.a**. On Red Hat systems, this library is provided by the **zlib-devel** rpm.

   If the library is found by Bacula during the **./configure** it will be mentioned in the **config.out** line by:

   ```
   ZLIB support:  yes
   ```

2. You must add the **compression=gzip** option on your Include statement in the Director's configuration file.

Bacula is Asking for a New Tape After 2 GB of Data but My Tape holds 33 GB. Why?
There are several reasons why Bacula will request a new tape.

- There is an I/O error on the tape. Bacula prints an error message and requests a new tape. Bacula does not attempt to continue writing after an I/O error.

- Bacula encounters and end of medium on the tape. This is not always distinguishable from an I/O error.
- You have specifically set some size limitation on the tape. For example the **Maximum Volume Bytes** or **Maximum Volume Files** in the Director's Pool resource, or **Maximum Volume Size** in the Storage daemon's Device resource.

34.28 Incremental backups are not working

Bacula is Not Doing the Right Thing When I Request an Incremental Backup. W

As explained in one of the previous questions, Bacula will automatically upgrade an Incremental or Differential job to a Full backup if it cannot find a prior Full backup or a suitable Full backup. For the gory details on how/when Bacula decides to upgrade levels please see the Level record in the Director's configuration chapter of this manual.

If after reading the above mentioned section, you believe that Bacula is not correctly handling the level (Differential/Incremental), please send us the following information for analysis:

- Your Director's configuration file.
- The output from **list jobs** covering the period where you are having the problem.
- The Job report output from the prior Full save (not critical).
- An **llist jobid=nnn** where nnn is the JobId of the prior Full save.
- The Job report output from the save that is doing the wrong thing (not critical).
- An **llist jobid=nnn** where nnn is the JobId of the job that was not correct.
- An explanation of what job went wrong and why you think it did.

The above information can allow us to analyze what happened, without it, there is not much we can do.

34.29 I am waiting forever for a backup of an offsite machine

I am Backing Up an Offsite Machine with an Unreliable Connection. The Director

Bacula was written on the assumption that it will have a good TCP/IP connection between all the daemons. As a consequence, the current Bacula doesn't deal with faulty connections very well. This situation is slowly being corrected over time.

There are several things you can do to improve the situation.

- Upgrade to version 1.32 and use the new SDConnectTimeout record. For example, set:

  ```
  SD Connect Timeout = 5 min
  ```

  in the FileDaemon resource.
- Run these kinds of jobs after all other jobs.

34.30 SSH hangs forever after starting Bacula

When I ssh into a machine and start Bacula then attempt to exit, ssh hangs forever.
This happens because Bacula leaves stdin, stdout, and stderr open for debug purposes. To avoid it, the simplest thing to do is to redirect the output of those files to **/dev/null** or another file in your startup script (the Red Hat autostart scripts do this automatically). For example, you start the Director with:

```
bacula-dir -c bacula-dir.conf ... >/dev/null 0>&1 2>\&1
```

and likewise for the other daemons.

34.31 I'm confused by retention periods

I'm confused by the different Retention periods: File Retention, Job Retention, Volume Re
Yes, this certainly can be confusing. The basic reason for so many is to allow flexibility. The File records take quite a lot of space in the catalog, so they are typically records you want to remove rather quickly. The Job records, take very little space, and they can be useful even without the File records to see what Jobs actually ran and when. One must understand that if the File records are removed from the catalog, you cannot use the **restore** command to restore an individual file since Bacula no longer knows where it is. However, as long as the Volume Retention period has not expired, the data will still be on the tape, and can be recovered from the tape.

For example, I keep a 30 day retention period for my Files to keep my catalog from getting too big, but I keep my tapes for a minimum of one year, just in case.

34.32 MaxVolumeSize is ignored

Why Does Bacula Ignore the MaxVolumeSize Set in my Pool?

The MaxVolumeSize that Bacula uses comes from the Media record, so most likely you changed your Pool, which is used as the default for creating Media records, **after** you created your Volume. Check what is in the Media record by doing:

```
llist Volume=xxx
```

If it doesn't have the right value, you can use:

```
update Volume=xxx
```

to change it.

34.33 I get a Connection refused when connecting to my Client

In connecting to my Client, I get "ERR:Connection Refused. Packet Size too big f

This is typically a communications error resulting from one of the following:

- Old versions of Bacula, usually a Win32 client, where two threads were using the same I/O packet. Fixed in more recent versions. Please upgrade.
- Some other program such as an HP Printer using the same port (9102 in this case).

If it is neither of the above, please submit a bug report at bugs.bacula.org.

Another solution might be to run the daemon with the debug option by:

```
Start a DOS shell Window.
cd c:\bacula\bin
bacula-fd -d100 -c c:\bacula\bin\bacula-fd.conf
```

This will cause the FD to write a file **bacula.trace** in the current directory, which you can examine to determine the problem.

34.34 Long running jobs die with Pipe Error

During long running jobs my File daemon dies with Pipe Error, or some other communicat There are a number of reasons why a connection might break. Most often, it is a router between your two computers that times out inactive lines (not respecting the keepalive feature that Bacula uses). In that case, you can use the **Heartbeat Interval** directive in both the Storage daemon and the File daemon.

In at least one case, the problem has been a bad driver for a Win32 NVidia NForce 3 ethernet card with driver (4.4.2 17/05/2004). In this case, a good driver is (4.8.2.0 06/04/2005). Moral of the story, make sure you have the latest ethernet drivers loaded, or use the following workaround as 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. For example, if you have:

```
iptables -t filter -A INPUT -m limit --limit 3/second --limit-burst 3 -j DROP
```

you will want to add the following rules **before** the above rule:

```
iptables -t filter -A INPUT --dport 9101 -j ACCEPT
iptables -t filter -A INPUT --dport 9102 -j ACCEPT
iptables -t filter -A INPUT --dport 9103 -j ACCEPT
```

This will ensure that any Bacula traffic will not get terminated because of high usage rates.

34.35 How do I tell the Job which Volume to use?

I can't figure out how to tell the job which volume to use This is an interesting statement. I now see that a number of people new to Bacula have the same problem as you, probably from using programs like tar.

In fact, you do not tell Bacula what tapes to use. It is the inverse. Bacula tells you want tapes it wants. You put tapes at its disposition and it chooses.

Now, if you *really* want to be tricky and try to tell Bacula what to do, it will be reasonable if for example you mount a valid tape that it can use on a drive, it will most likely go ahead and use it. It also has a documented algorithm for choosing tapes – but you are asking for problems ...

So, the trick is to invert your concept of things and put Bacula in charge of handling the tapes. Once you do that, you will be fine. If you want to anticipate what it is going to do, you can generally figure it out correctly and get what you want.

If you start with the idea that you are going to force or tell Bacula to use particular tapes or you insist on trying to run in that kind of mode, you will probably not be too happy.

I don't want to worry about what tape has what data. That is what Bacula is designed for.

If you have an application where you *really* need to remove a tape each day and insert a new one, it can be done the directives exist to accomplish that. In such a case, one little "trick" to knowing what tape Bacula will want at 2am while you are asleep is to run a tiny job at 4pm while you are still at work that backs up say one directory, or even one file. You will quickly find out what tape it wants, and you can mount it before you go home ...

34.36 Password generation

How do I generate a password? Each daemon needs a password. This password occurs in the configuration file for that daemon and in the bacula-dir.conf file. These passwords are plain text. There is no special generation procedure. Most people just use random text.

Passwords are never sent over the wire in plain text. They are always encrypted.

Security surrounding these passwords is best left security to your operating system. Passwords are not encrypted within Bacula configuration files.

Chapter 35

Tips and Suggestions

There are a number of example scripts for various things that can be found in the **example** subdirectory and its subdirectories of the Bacula source distribution.

For additional tips, please see the Bacula wiki.

35.1 Upgrading Bacula Versions

The first thing to do before upgrading from one version to another is to ensure that you don't overwrite or delete your production (current) version of Bacula until you have tested that the new version works.

If you have installed Bacula into a single directory, this is simple: simply make a copy of your Bacula directory.

If you have done a more typical Unix installation where the binaries are placed in one directory and the configuration files are placed in another, then the simplest way is to configure your new Bacula to go into a single file. Alternatively, make copies of all your binaries and especially your conf files.

Whatever your situation may be (one of the two just described), you should probably start with the **defaultconf** script that can be found in the **examples** subdirectory. Copy this script to the main Bacula directory, modify it as necessary (there should not need to be many modifications), configure Bacula, build it, install it, then stop your production Bacula, copy all the ***.conf** files from your production Bacula directory to the test Bacula directory, start the test version, and run a few test backups. If all seems good, then you can proceed to install the new Bacula in place of or possibly over the old Bacula.

When installing a new Bacula you need not worry about losing the changes you made to your configuration files as the installation process will not overwrite them providing that you do not do a **make uninstall**.

If the new version of Bacula requires an upgrade to the database, you can upgrade it with the script **update_bacula_tables**, which will be installed in your scripts directory (default **/etc/bacula**), or alternatively, you can find it in the <**bacula-source**>**/src/cats** directory.

35.2 Getting Notified of Job Completion

One of the first things you should do is to ensure that you are being properly notified of the status of each Job run by Bacula, or at a minimum of each Job that terminates with an error.

Until you are completely comfortable with **Bacula**, we recommend that you send an email to yourself for each Job that is run. This is most easily accomplished by adding an email notification address in the **Messages** resource of your Director's configuration file. An email is automatically configured in the default configuration files, but you must ensure that the default **root** address is replaced by your email address.

For additional examples of how to configure a Bacula, please take a look at the **.conf** files found in the **examples** sub-directory. We recommend the following configuration (where you change the paths and email address to correspond to your setup). Note, the **mailcommand** and **operatorcommand** should be on a single line. They were split here for presentation:

```
Messages {
  Name = Standard
  mailcommand = "/home/bacula/bin/bsmtp -h localhost
              -f \"\(Bacula\) %r\"
              -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/home/bacula/bin/bsmtp -h localhost
              -f \"\(Bacula\) %r\"
              -s \"Bacula: Intervention needed for %j\" %r"
  Mail = your-email-address = all, !skipped, !terminate
  append = "/home/bacula/bin/log" = all, !skipped, !terminate
  operator = your-email-address = mount
  console = all, !skipped, !saved
}
```

You will need to ensure that the **/home/bacula/bin** path on the **mailcommand** and the **operatorcommand** lines point to your **Bacula** binary directory where the **bsmtp** program will be installed. You will also want to ensure that the **your-email-address** is replaced by your email address, and finally, you will also need to ensure that the **/home/bacula/bin/log** points to the file where you want to log all messages.

With the above Messages resource, you will be notified by email of every Job that ran, all the output will be appended to the **log** file you specify, all output will be directed to the console program, and all mount messages will be emailed to you. Note, some messages will be sent to multiple destinations.

The form of the mailcommand is a bit complicated, but it allows you to distinguish whether the Job terminated in error or terminated normally. Please see the Mail Command section of the Messages Resource chapter of this manual for the details of the substitution characters used above.

Once you are totally comfortable with Bacula as I am, or if you have a large number of nightly Jobs as I do (eight), you will probably want to change the **Mail** command to **Mail On Error** which will generate an email message only if the Job terminates in error. If the Job terminates normally, no email message will be sent, but the output will still be appended to the log file as well as sent to the Console program.

35.3 Getting Email Notification to Work

The section above describes how to get email notification of job status. Occasionally, however, users have problems receiving any email at all. In that case, the things to check are the following:

- Ensure that you have a valid email address specified on your **Mail** record in the Director's Messages resource. The email address should be fully qualified. Simply using **root** generally will not work, rather you should use **root@localhost** or better yet your full domain.
- Ensure that you do not have a **Mail** record in the Storage daemon's or File daemon's configuration files. The only record you should have is **director**:

```
director = director-name = all
```

- If all else fails, try replacing the **mailcommand** with

```
mailcommand = "mail -s test your@domain.com"
```

- Once the above is working, assuming you want to use **bsmtp**, submit the desired bsmtp command by hand and ensure that the email is delivered, then put that command into **Bacula**. Small differences in things such as the parenthesis around the word Bacula can make a big difference to some bsmtp programs. For example, you might start simply by using:

```
mailcommand = "/home/bacula/bin/bsmtp -f \"root@localhost\" %r"
```

35.4 Getting Notified that Bacula is Running

If like me, you have setup Bacula so that email is sent only when a Job has errors, as described in the previous section of this chapter, inevitably, one day, something will go wrong and **Bacula** can stall. This could be because Bacula crashes, which is vary rare, or more likely the network has caused **Bacula** to **hang** for some unknown reason.

To avoid this, you can use the **RunAfterJob** command in the Job resource to schedule a Job nightly, or weekly that simply emails you a message saying that Bacula is still running. For example, I have setup the following Job in my Director's configuration file:

```
Schedule {
  Name = "Watchdog"
  Run = Level=Full sun-sat at 6:05
}
Job {
  Name = "Watchdog"
  Type = Admin
  Client=Watchdog
  FileSet="Verify Set"
  Messages = Standard
  Storage = DLTDrive
  Pool = Default
  Schedule = "Watchdog"
  RunAfterJob = "/home/kern/bacula/bin/watchdog %c %d"
}
Client {
  Name = Watchdog
  Address = rufus
  FDPort = 9102
  Catalog = Verify
  Password = ""
  File Retention = 1day
```

```
  Job Retention = 1 month
  AutoPrune = yes
}
```

Where I established a schedule to run the Job nightly. The Job itself is type **Admin** which means that it doesn't actually do anything, and I've defined a FileSet, Pool, Storage, and Client, all of which are not really used (and probably don't need to be specified). The key aspect of this Job is the command:

```
  RunAfterJob = "/home/kern/bacula/bin/watchdog %c %d"
```

which runs my "watchdog" script. As an example, I have added the Job codes %c and %d which will cause the Client name and the Director's name to be passed to the script. For example, if the Client's name is **Watchdog** and the Director's name is **main-dir** then referencing $1 in the script would get **Watchdog** and referencing $2 would get **main-dir**. In this case, having the script know the Client and Director's name is not really useful, but in other situations it may be.

You can put anything in the watchdog script. In my case, I like to monitor the size of my catalog to be sure that **Bacula** is really pruning it. The following is my watchdog script:

```
#!/bin/sh
cd /home/kern/mysql/var/bacula
du . * |
/home/kern/bacula/bin/bsmtp  \
   -f "\(Bacula\) abuse@whitehouse.com" -h mail.yyyy.com \
   -s "Bacula running" abuse@whitehouse.com
```

If you just wish to send yourself a message, you can do it with:

```
#!/bin/sh
cd /home/kern/mysql/var/bacula
/home/kern/bacula/bin/bsmtp  \
   -f "\(Bacula\) abuse@whitehouse.com" -h mail.yyyy.com \
   -s "Bacula running" abuse@whitehouse.com <<END-OF-DATA
Bacula is still running!!!
END-OF-DATA
```

35.5 Maintaining a Valid Bootstrap File

By using a WriteBootstrap record in each of your Director's Job resources, you can constantly maintain a bootstrap file that will enable you to recover the state of your system as of the last backup without having the Bacula catalog. This permits you to more easily recover from a disaster that destroys your Bacula catalog.

When a Job resource has a **WriteBootstrap** record, Bacula will maintain the designated file (normally on another system but mounted by NSF) with up to date information necessary to restore your system. For example, in my Director's configuration file, I have the following record:

```
Write Bootstrap = "/mnt/deuter/files/backup/client-name.bsr"
```

where I replace **client-name** by the actual name of the client that is being backed up. Thus, Bacula automatically maintains one file for each of my clients. The necessary bootstrap information is appended to this file during each **Incremental** backup, and the file is totally rewritten during each **Full** backup.

Note, one disadvantage of writing to an NFS mounted volume as I do is that if the other machine goes down, the OS will wait forever on the fopen() call that Bacula makes. As a consequence, Bacula will completely stall until the machine exporting the NFS mounts comes back up. A possible solution to this problem was provided by Andrew Hilborne, and consists of using the **soft** option instead of the **hard** option when mounting the NFS volume, which is typically done in **/etc/fstab**/. The NFS documentation explains these options in detail. However, I found that with the **soft** option NFS disconnected frequently causing even more problems.

If you are starting off in the middle of a cycle (i.e. with Incremental backups) rather than at the beginning (with a Full backup), the **bootstrap** file will not be immediately valid as it must always have the information from a Full backup as the first record. If you wish to synchronize your bootstrap file immediately, you can do so by running a **restore** command for the client and selecting a full restore, but when the restore command asks for confirmation to run the restore Job, you simply reply no, then copy the bootstrap file that was written to the location specified on the **Write Bootstrap** record. The restore bootstrap file can be found in **restore.bsr** in the working directory that you defined. In the example given below for the client **rufus**, my input is shown in bold. Note, the JobId output has been partially truncated to fit on the page here:

```
(in the Console program)
*restore
First you select one or more JobIds that contain files
to be restored. You will then 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 JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Cancel
Select item:  (1-6): 5
The defined Client resources are:
     1: Minimatou
     2: Rufus
     3: Timmy
Select Client (File daemon) resource (1-3): 2
The defined FileSet resources are:
     1: Other Files
Item 1 selected automatically.
+-------+------+-------+---------+---------+------+-------+------------+
| JobId | Levl | Files | StrtTim | VolName | File | SesId | VolSesTime |
+-------+------+-------+---------+---------+------+-------+------------+
| 2     | F    | 84    |  ...    | test1   | 0    | 1     | 1035645259 |
+-------+------+-------+---------+---------+------+-------+------------+
You have selected the following JobId: 2
Building directory tree for JobId 2 ...
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: /
$ done
84 files selected to restore.
Run Restore job
JobName:    kernsrestore
Bootstrap:  /home/kern/bacula/working/restore.bsr
Where:      /tmp/bacula-restores
FileSet:    Other Files
Client:     Rufus
Storage:    File
JobId:      *None*
OK to run? (yes/mod/no): no
quit
(in a shell window)
cp ../working/restore.bsr /mnt/deuter/files/backup/rufus.bsr
```

35.6 Rejected Volumes After a Crash

Bacula keeps a count of the number of files on each Volume in its Catalog database so that before appending to a tape, it can verify that the number of files are correct, and thus prevent overwriting valid data. If the Director or the Storage daemon crashes before the job has completed, the tape will contain one more file than is noted in the Catalog, and the next time you attempt to use the same Volume, Bacula will reject it due to a mismatch between the physical tape (Volume) and the catalog.

The easiest solution to this problem is to label a new tape and start fresh. If you wish to continue appending to the current tape, you can do so by using the **update** command in the console program to change the **Volume Files** entry in the catalog. A typical sequence of events would go like the following:

```
- Bacula crashes
- You restart Bacula
```

Bacula then prints:

```
17-Jan-2003 16:45 rufus-dir: Start Backup JobId 13,
                  Job=kernsave.2003-01-17_16.45.46
17-Jan-2003 16:45 rufus-sd: Volume test01 previously written,
                  moving to end of data.
17-Jan-2003 16:46 rufus-sd: kernsave.2003-01-17_16.45.46 Error:
                  I cannot write on this volume because:
                  The number of files mismatch! Volume=11 Catalog=10
17-Jan-2003 16:46 rufus-sd: Job kernsave.2003-01-17_16.45.46 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
```

(note, lines wrapped for presentation) The key here is the line that reads:

```
  The number of files mismatch! Volume=11 Catalog=10
```

It says that Bacula found eleven files on the volume, but that the catalog says there should be ten. When you see this, you can be reasonably sure that the SD was interrupted while writing before it had a chance to update the catalog. As a consequence, you can just modify the catalog count to eleven, and even if the catalog contains references to files saved in file 11, everything will be OK and nothing will be lost. Note that if the SD had written several file marks to the volume, the difference between the Volume count and the Catalog count could be larger than one, but this is unusual.

If on the other hand the catalog is marked as having more files than Bacula found on the tape, you need to consider the possible negative consequences of modifying the catalog. Please see below for a more complete discussion of this.

Continuing with the example of **Volume = 11 Catalog = 10**, to enable to Bacula to append to the tape, you do the following:

```
update
Update choice:
     1: Volume parameters
     2: Pool from resource
     3: Slots from autochanger
Choose catalog item to update (1-3): 1
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2):
+-------+---------+--------+---------+-----------+------+----------+------+-----+
| MedId | VolName | MedTyp | VolStat | VolBytes  | Last | VolReten | Recy | Slt |
+-------+---------+--------+---------+-----------+------+----------+------+-----+
| 1     | test01  | DDS-4  | Error   | 352427156 | ...  | 31536000 | 1    | 0   |
+-------+---------+--------+---------+-----------+------+----------+------+-----+
Enter MediaId or Volume name: 1
```

(note table output truncated for presentation) First, you chose to update the Volume parameters by entering a **1**. In the volume listing that follows, notice how the VolStatus is **Error**. We will correct that after changing the Volume Files. Continuing, you respond 1,

```
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Pool
    11: Done
```

```
Select parameter to modify (1-11): 9
Warning changing Volume Files can result
in loss of data on your Volume
Current Volume Files is: 10
Enter new number of Files for Volume: 11
New Volume Files is: 11
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Pool
    11: Done
Select parameter to modify (1-10): 1
```

Here, you have selected **9** in order to update the Volume Files, then you changed it from **10** to **11**, and you now answer **1** to change the Volume Status.

```
Current Volume status is: Error
Possible Values are:
     1: Append
     2: Archive
     3: Disabled
     4: Full
     5: Used
     6: Read-Only
Choose new Volume Status (1-6): 1
New Volume status is: Append
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Pool
    11: Done
Select parameter to modify (1-11): 11
Selection done.
```

At this point, you have changed the Volume Files from **10** to **11** to account for the last file that was written but not updated in the database, and you changed the Volume Status back to **Append**.

This was a lot of words to describe something quite simple.

The **Volume Files** option exists only in version 1.29 and later, and you should be careful using it. Generally, if you set the value to that which Bacula said is on the tape, you will be OK, especially if the value is one more than what is in the catalog.

Now lets consider the case:

```
The number of files mismatch! Volume=10 Catalog=12
```

Here the Bacula found fewer files on the volume than what is marked in the catalog. Now, in this case, you should hesitate a lot before modifying the count in the catalog, because if you force the catalog from 12 to 10, Bacula will start writing after the file 10 on the tape, possibly overwriting valid data, and if you ever try to restore any of the files that the catalog has marked as saved on Files 11 and 12, all chaos will break out. In this case, you will probably be better off using a new tape. In fact, you might want to see what files the catalog claims are actually stored on that Volume, and back them up to another tape and recycle this tape.

35.7 Security Considerations

Only the File daemon needs to run with root permission (so that it can access all files). As a consequence, you may run your Director, Storage daemon, and MySQL or PostgreSQL database server as non-root processes. Version 1.30 has the **-u** and the **-g** options that allow you to specify a userid and groupid on the command line to be used after Bacula starts.

As of version 1.33, thanks to Dan Langille, it is easier to configure the Bacula Director and Storage daemon to run as non-root.

You should protect the Bacula port addresses (normally 9101, 9102, and 9103) from outside access by a firewall or other means of protection to prevent unauthorized use of your daemons.

You should ensure that the configuration files are not world readable since they contain passwords that allow access to the daemons. Anyone who can access the Director using a console program can restore any file from a backup Volume.

You should protect your Catalog database. If you are using SQLite, make sure that the working directory is readable only by root (or your Bacula userid), and ensure that **bacula.db** has permissions **-rw-r--r--** (i.e. 640) or more strict. If you are using MySQL or PostgreSQL, please note that the Bacula setup procedure leaves the database open to anyone. At a minimum, you should assign the user **bacula** a userid and add it to your Director's configuration file in the appropriate Catalog resource.

If you use the make_catalog_backup script provided by Bacula, remember that you should take care when supplying passwords on the command line. Read the Backing Up Your Bacula Database - Security Considerations section for more information.

35.8 Creating Holiday Schedules

If you normally change tapes every day or at least every Friday, but Thursday is a holiday, you can use a trick proposed by Lutz Kittler to ensure that no job runs on Thursday so that you can insert Friday's tape and be sure it will be used on Friday. To do so, define a **RunJobBefore** script that normally returns zero, so that the Bacula job will normally continue. You can then modify the script to return non-zero on any day when you do not want Bacula to run the job.

35.9 Automatic Labeling Using Your Autochanger

If you have an autochanger but it does not support barcodes, using a "trick" you can make Bacula automatically label all the volumes in your autochanger's magazine.

First create a file containing one line for each slot in your autochanger that has a tape to be labeled. The line will contain the slot number a colon (:) then the Volume name you want to use. For example, create a file named **volume-list**, which contains:

```
1:Volume001
2:TestVolume02
5:LastVolume
```

The records do not need to be in any order and you don't need to mention all the slots. Normally, you will have a consistent set of Volume names and a sequential set of numbers for each slot you want labeled. In the example above, I've left out slots 3 and 4 just as an example. Now, modify your **mtx-changer** script and comment out all the lines in the **list)** case by putting a # in column 1. Then add the following two lines:

```
  cat <absolute-path>/volume-list
  exit 0
```

so that the whole case looks like:

```
  list)
#
# commented out lines
   cat <absolute-path>/volume-list
   exit 0
   ;;
```

where you replace <absolute-path> with the full path to the volume-list file. Then using the console, you enter the following command:

```
   label barcodes
```

and Bacula will proceed to mount the autochanger Volumes in the list and label them with the Volume names you have supplied. Bacula will think that the list was provided by the autochanger barcodes, but in reality, it was you who supplied the <barcodes>.

If it seems to work, when it finishes, enter:

```
   list volumes
```

and you should see all the volumes nicely created.

35.10 Backing Up Portables Using DHCP

You may want to backup laptops or portables that are not always connected to the network. If you are using DHCP to assign an IP address to those machines when they connect, you will need to use the Dynamic Update capability of DNS to assign a name to those machines that can be used in the Address field of the Client resource in the Director's conf file.

35.11 Going on Vacation

At some point, you may want to be absent for a week or two and you want to make sure Bacula has enough tape left so that the backups will complete. You start by doing a **list volumes** in the Console program:

```
list volumes

Using default Catalog name=BackupDB DB=bacula
Pool: Default
+---------+---------------+-----------+-----------+----------------+-
| MediaId | VolumeName    | MediaType | VolStatus |       VolBytes |
+---------+---------------+-----------+-----------+----------------+-
|      23 | DLT-30Nov02   | DLT8000   | Full      | 54,739,278,128 |
|      24 | DLT-21Dec02   | DLT8000   | Full      | 56,331,524,629 |
|      25 | DLT-11Jan03   | DLT8000   | Full      | 67,863,514,895 |
|      26 | DLT-02Feb03   | DLT8000   | Full      | 63,439,314,216 |
|      27 | DLT-03Mar03   | DLT8000   | Full      | 66,022,754,598 |
|      28 | DLT-04Apr03   | DLT8000   | Full      | 60,792,559,924 |
|      29 | DLT-28Apr03   | DLT8000   | Full      | 62,072,494,063 |
|      30 | DLT-17May03   | DLT8000   | Full      | 65,901,767,839 |
|      31 | DLT-07Jun03   | DLT8000   | Used      | 56,558,490,015 |
|      32 | DLT-28Jun03   | DLT8000   | Full      | 64,274,871,265 |
|      33 | DLT-19Jul03   | DLT8000   | Full      | 64,648,749,480 |
|      34 | DLT-08Aug03   | DLT8000   | Full      | 64,293,941,255 |
|      35 | DLT-24Aug03   | DLT8000   | Append    |  9,999,216,782 |
+---------+---------------+-----------+-----------+----------------+
```

Note, I have truncated the output for presentation purposes. What is significant, is that I can see that my current tape has almost 10 Gbytes of data, and that the average amount of data I get on my tapes is about 60 Gbytes. So if I go on vacation now, I don't need to worry about tape capacity (at least not for short absences).

Equally significant is the fact that I did go on vacation the 28th of June 2003, and when I did the **list volumes** command, my current tape at that time, DLT-07Jun03 MediaId 31, had 56.5 Gbytes written. I could see that the tape would fill shortly. Consequently, I manually marked it as **Used** and replaced it with a fresh tape that I labeled as DLT-28Jun03, thus assuring myself that the backups would all complete without my intervention.

35.12 Exclude Files on Windows Regardless of Case

This tip was submitted by Marc Brueckner who wasn't sure of the case of some of his files on Win32, which is case insensitive. The problem is that Bacula thinks that **/UNIMPORTANT FILES** is different from **/Unimportant Files**. Marc was aware that the file exclusion permits wild-cards. So, he specified:

```
"/[Uu][Nn][Ii][Mm][Pp][Oo][Rr][Tt][Aa][Nn][Tt]  [Ff][Ii][Ll][Ee][Ss]"
```

As a consequence, the above exclude works for files of any case.

Please note that this works only in Bacula Exclude statement and not in Include.

35.13 Executing Scripts on a Remote Machine

This tip also comes from Marc Brueckner. (Note, this tip is probably outdated by the addition of **ClientRunBeforJob** and **ClientRunAfterJob** Job records, but the technique still could be useful.) First I thought the "Run Before Job" statement in the Job-resource is for executing a script on the remote machine (the machine to be backed up). (Note, this is possible as mentioned above by using **ClientRunBeforJob** and **ClientRunAfterJob**). It could be useful to execute scripts on the remote machine e.g. for stopping databases or other services while doing the backup. (Of course I have to start the services again when the backup has finished) I found the following solution: Bacula could execute scripts on the remote machine by using ssh. The authentication is done automatically using a private key. First you have to generate a keypair. I've done this by:

```
ssh-keygen -b 4096 -t dsa -f Bacula_key
```

This statement may take a little time to run. It creates a public/private key pair with no passphrase. You could save the keys in /etc/bacula. Now you have two new files : Bacula_key which contains the private key and Bacula_key.pub which contains the public key.

Now you have to append the Bacula_key.pub file to the file authorized_keys in the \root\.ssh directory of the remote machine. Then you have to add (or uncomment) the line

```
AuthorizedKeysFile          %h/.ssh/authorized_keys
```

to the sshd_config file on the remote machine. Where the %h stands for the home-directory of the user (root in this case).

Assuming that your sshd is already running on the remote machine, you can now enter the following on the machine where Bacula runs:

```
ssh -i Bacula_key  -l root <machine-name-or-ip-address> "ls -la"
```

This should execute the "ls -la" command on the remote machine.

Now you could add lines like the following to your Director's conf file:

```
...
Run Before Job = ssh -i /etc/bacula/Bacula_key 192.168.1.1 \
                 "/etc/init.d/database stop"
Run After Job = ssh -i /etc/bacula/Bacula_key 192.168.1.1 \
                 "/etc/init.d/database start"
...
```

Even though Bacula version 1.32 and later has a ClientRunBeforeJob, the ssh method still could be useful for updating all the Bacula clients on several remote machines in a single script.

35.14 Recycling All Your Volumes

This tip comes from Phil Stracchino.

If you decide to blow away your catalog and start over, the simplest way to re-add all your prelabeled tapes with a minimum of fuss (provided you don't care about the data on the tapes) is to add the tape labels using the console **add** command, then go into the catalog and manually set the VolStatus of every tape to **Recycle**.

The SQL command to do this is very simple, either use your vendor's command line interface (mysql, postgres, sqlite, ...) or use the sql command in the Bacula console:

```
update Media set VolStatus='Recycle';
```

Bacula will then ignore the data already stored on the tapes and just re-use each tape without further objection.

35.15 Backing up ACLs on ext3 or XFS filesystems

This tip comes from Volker Sauer.

Note, this tip was given prior to implementation of ACLs in Bacula (version 1.34.5). It is left here because dumping/displaying ACLs can still be useful in testing/verifying that Bacula is backing up and restoring your ACLs properly. Please see the aclsupport FileSet option in the configuration chapter of this manual.

For example, you could dump the ACLs to a file with a script similar to the following:

```
#!/bin/sh
BACKUP_DIRS="/foo /bar"
STORE_ACL=/root/acl-backup
umask 077
for i in $BACKUP_DIRS; do
 cd $i /usr/bin/getfacl -R --skip-base .>$STORE_ACL/${i//\//_}
done
```

Then use Bacula to backup **/root/acl-backup**.

The ACLs could be restored using Bacula to the **/root/acl-backup** file, then restored to your system using:

```
setfacl --restore/root/acl-backup
```

35.16 Total Automation of Bacula Tape Handling

This tip was provided by Alexander Kuehn.

Bacula is a really nice backup program except that the manual tape changing requires user interaction with the bacula console.

Fortunately I can fix this. NOTE!!! This suggestion applies for people who do *NOT* have tape autochangers and must change tapes manually.!!!!!

Bacula supports a variety of tape changers through the use of mtx-changer scripts/programs. This highly flexible approach allowed me to create this shell script which does the following: Whenever a new tape is required it sends a mail to the operator to insert the new tape. Then it waits until a tape has been inserted, sends a mail again to say thank you and let's bacula continue its backup. So you can schedule and run backups without ever having to log on or see the console. To make the whole thing work you need to create a Device resource which looks something like this ("Archive Device", "Maximum Changer Wait", "Media Type" and "Label media" may have different values):

```
Device {
   Name=DDS3
   Archive Device = # use yours not mine! ;)/dev/nsa0
   Changer Device = # not really required/dev/nsa0
   Changer Command = "# use this (maybe change the path)!
         /usr/local/bin/mtx-changer %o %a %S"
   Maximum Changer Wait = 3d          # 3 days in seconds
   AutomaticMount = yes;              # mount on start
   AlwaysOpen = yes;                  # keep device locked
   Media Type = DDS3                  # it's just a name
   RemovableMedia = yes;              #
   Offline On Unmount = Yes;          # keep this too
   Label media = Yes;                 #
}
```

As the script has to emulate the complete wisdom of a mtx-changer it has an internal "database" containing where which tape is stored, you can see this on the following line:

```
labels="VOL-0001 VOL-0002 VOL-0003 VOL-0004 VOL-0005 VOL-0006
VOL-0007 VOL-0008 VOL-0009 VOL-0010 VOL-0011 VOL-0012"
```

The above should be all on one line, and it effectively tells Bacula that volume "VOL-0001" is located in slot 1 (which is our lowest slot), that volume "VOL-0002" is located in slot 2 and so on.. The script also maintains a logfile (/var/log/mtx.log) where you can monitor its operation.

35.17 Running Concurrent Jobs

Bacula can run multiple concurrent jobs, but the default configuration files do not enable it. Using the **Maximum Concurrent Jobs** directive, you can configure how many and which jobs can be run simultaneously. The Director's default value for **Maximum Concurrent Jobs** is "1".

To initially setup concurrent jobs you need to define **Maximum Concurrent Jobs** in the Director's configuration file (bacula-dir.conf) in the Director, Job, Client, and Storage resources.

Additionally the File daemon, and the Storage daemon each have their own **Maximum Concurrent Jobs** directive that sets the overall maximum number of concurrent jobs the daemon will run. The default for both the File daemon and the Storage daemon is "20".

For example, if you want two different jobs to run simultaneously backing up the same Client to the same Storage device, they will run concurrently only if you have set **Maximum Concurrent Jobs** greater than one in the Director resource, the Client resource, and the Storage resource in bacula-dir.conf.

We recommend that you read the Data Spooling of this manual first, then test your multiple concurrent backup including restore testing before you put it into production.

Below is a super stripped down bacula-dir.conf file showing you the four places where the the file must be modified to allow the same job **NightlySave** to run up to four times concurrently. The change to the Job resource is not necessary if you want different Jobs to run at the same time, which is the normal case.

```
#
# Bacula Director Configuration file -- bacula-dir.conf
#
Director {
  Name = rufus-dir
  Maximum Concurrent Jobs = 4
  ...
}
Job {
  Name = "NightlySave"
  Maximum Concurrent Jobs = 4
  Client = rufus-fd
  Storage = File
  ...
}
Client {
  Name = rufus-fd
  Maximum Concurrent Jobs = 4
  ...
}
Storage {
  Name = File
  Maximum Concurrent Jobs = 4
  ...
}
```

Chapter 36

Volume Utility Tools

This document describes the utility programs written to aid Bacula users and developers in dealing with Volumes external to Bacula.

36.1 Specifying the Configuration File

Starting with version 1.27, each of the following programs requires a valid Storage daemon configuration file (actually, the only part of the configuration file that these programs need is the **Device** resource definitions). This permits the programs to find the configuration parameters for your archive device (generally a tape drive). By default, they read **bacula-sd.conf** in the current directory, but you may specify a different configuration file using the **-c** option.

36.2 Specifying a Device Name For a Tape

Each of these programs require a **device-name** where the Volume can be found. In the case of a tape, this is the physical device name such as **/dev/nst0** or **/dev/rmt/0ubn** depending on your system. For the program to work, it must find the identical name in the Device resource of the configuration file. See below for specifying Volume names.

Please note that if you have Bacula running and you ant to use one of these programs, you will either need to stop the Storage daemon, or **unmount** any tape drive you want to use, otherwise the drive will **busy** because Bacula is using it.

36.3 Specifying a Device Name For a File

If you are attempting to read or write an archive file rather than a tape, the **device-name** should be the full path to the archive location including the filename. The filename (last part of the specification) will be stripped and used as the Volume name, and the path (first part before the filename) must have the same entry in the configuration file. So, the path is equivalent to the archive device name, and the filename is equivalent to the volume name.

36.4 Specifying Volumes

In general, you must specify the Volume name to each of the programs below (with the exception of **btape**). The best method to do so is to specify a **bootstrap** file on the command line with the **-b** option. As part of the bootstrap file, you will then specify the Volume name or Volume names if more than one volume is needed. For example, suppose you want to read tapes **tape1** and **tape2**. First construct a **bootstrap** file named say, **list.bsr** which contains:

```
Volume=test1|test2
```

where each Volume is separated by a vertical bar. Then simply use:

```
./bls -b list.bsr /dev/nst0
```

In the case of Bacula Volumes that are on files, you may simply append volumes as follows:

```
./bls /tmp/test1\|test2
```

where the backslash (\) was necessary as a shell escape to permit entering the vertical bar (—).

And finally, if you feel that specifying a Volume name is a bit complicated with a bootstrap file, you can use the **-V** option (on all programs except **bcopy**) to specify one or more Volume names separated by the vertical bar (—). For example,

```
./bls -V Vol001 /dev/nst0
```

You may also specify an asterisk (*) to indicate that the program should accept any volume. For example:

```
./bls -V* /dev/nst0
```

36.5 bls

bls can be used to do an **ls** type listing of a **Bacula** tape or file. It is called:

```
Usage: bls [options] <device-name>
       -b <file>       specify a bootstrap file
       -c <file>       specify a config file
       -d <level>      specify debug level
       -e <file>       exclude list
       -i <file>       include list
       -j              list jobs
       -k              list blocks
    (no j or k option) list saved files
       -L              dump label
       -p              proceed inspite of errors
       -v              be verbose
       -V              specify Volume names (separated by |)
       -?              print this message
```

For example, to list the contents of a tape:

```
./bls -V Volume-name /dev/nst0
```

Or to list the contents of a file:

```
./bls /tmp/Volume-name
or
./bls -V Volume-name /tmp
```

Note that, in the case of a file, the Volume name becomes the filename, so in the above example, you will replace the **xxx** with the name of the volume (file) you wrote.

Normally if no options are specified, **bls** will produce the equivalent output to the **ls -l** command for each file on the tape. Using other options listed above, it is possible to display only the Job records, only the tape blocks, etc. For example:

```
./bls /tmp/File002
bls: butil.c:148 Using device: /tmp
drwxrwxr-x   3 k  k  4096 02-10-19 21:08  /home/kern/bacula/k/src/dird/
drwxrwxr-x   2 k  k  4096 02-10-10 18:59  /home/kern/bacula/k/src/dird/CVS/
-rw-rw-r--   1 k  k    54 02-07-06 18:02  /home/kern/bacula/k/src/dird/CVS/Root
-rw-rw-r--   1 k  k    16 02-07-06 18:02  /home/kern/bacula/k/src/dird/CVS/Repository
-rw-rw-r--   1 k  k  1783 02-10-10 18:59  /home/kern/bacula/k/src/dird/CVS/Entries
-rw-rw-r--   1 k  k 97506 02-10-18 21:07  /home/kern/bacula/k/src/dird/Makefile
-rw-r--r--   1 k  k  3513 02-10-18 21:02  /home/kern/bacula/k/src/dird/Makefile.in
-rw-rw-r--   1 k  k  4669 02-07-06 18:02  /home/kern/bacula/k/src/dird/README-config
-rw-r--r--   1 k  k  4391 02-09-14 16:51  /home/kern/bacula/k/src/dird/authenticate.c
-rw-r--r--   1 k  k  3609 02-07-07 16:41  /home/kern/bacula/k/src/dird/autoprune.c
-rw-rw-r--   1 k  k  4418 02-10-18 21:03  /home/kern/bacula/k/src/dird/bacula-dir.conf
...
-rw-rw-r--   1 k  k    83 02-08-31 19:19  /home/kern/bacula/k/src/dird/.cvsignore
bls: Got EOF on device /tmp
84 files found.
```

36.5.1 Listing Jobs

If you are listing a Volume to determine what Jobs to restore, normally the **-j** option provides you with most of what you will need as long as you don't have multiple clients. For example,

```
./bls -j -V Test1 -c stored.conf DDS-4
bls: butil.c:258 Using device: "DDS-4" for reading.
11-Jul 11:54 bls: Ready to read from volume "Test1" on device "DDS-4" (/dev/nst0).
Volume Record: File:blk=0:1 SessId=4 SessTime=1121074625 JobId=0 DataLen=165
Begin Job Session Record: File:blk=0:2 SessId=4 SessTime=1121074625 JobId=1 Level=F Type=B
Begin Job Session Record: File:blk=0:3 SessId=5 SessTime=1121074625 JobId=5 Level=F Type=B
Begin Job Session Record: File:blk=0:6 SessId=3 SessTime=1121074625 JobId=2 Level=F Type=B
Begin Job Session Record: File:blk=0:13 SessId=2 SessTime=1121074625 JobId=4 Level=F Type=B
End Job Session Record: File:blk=0:99 SessId=3 SessTime=1121074625 JobId=2 Level=F Type=B
   Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:101 SessId=2 SessTime=1121074625 JobId=4 Level=F Type=B
   Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:108 SessId=5 SessTime=1121074625 JobId=5 Level=F Type=B
   Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:109 SessId=4 SessTime=1121074625 JobId=1 Level=F Type=B
   Files=168 Bytes=1,732,978 Errors=0 Status=T
11-Jul 11:54 bls: End of Volume at file 1 on device "DDS-4" (/dev/nst0), Volume "Test1"
11-Jul 11:54 bls: End of all volumes.
```

shows a full save followed by two incremental saves.

Adding the **-v** option will display virtually all information that is available for each record:

36.5.2 Listing Blocks

Normally, except for debugging purposes, you will not need to list Bacula blocks (the "primitive" unit of Bacula data on the Volume). However, you can do so with:

```
./bls -k /tmp/File002
bls: butil.c:148 Using device: /tmp
Block: 1 size=64512
Block: 2 size=64512
...
Block: 65 size=64512
Block: 66 size=19195
bls: Got EOF on device /tmp
End of File on device
```

By adding the **-v** option, you can get more information, which can be useful in knowing what sessions were written to the volume:

```
./bls -k -v /tmp/File002
Volume Label:
Id                : Bacula 0.9 mortal
VerNo             : 10
VolName           : File002
PrevVolName       :
VolFile           : 0
LabelType         : VOL_LABEL
LabelSize         : 147
PoolName          : Default
MediaType         : File
PoolType          : Backup
HostName          :
Date label written: 2002-10-19 at 21:16
Block: 1 blen=64512 First rec FI=VOL_LABEL SessId=1 SessTim=1035062102 Strm=0 rlen=147
Block: 2 blen=64512 First rec FI=6 SessId=1 SessTim=1035062102 Strm=DATA rlen=4087
Block: 3 blen=64512 First rec FI=12 SessId=1 SessTim=1035062102 Strm=DATA rlen=5902
Block: 4 blen=64512 First rec FI=19 SessId=1 SessTim=1035062102 Strm=DATA rlen=28382
...
Block: 65 blen=64512 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=1873
Block: 66 blen=19195 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=2973
bls: Got EOF on device /tmp
End of File on device
```

Armed with the SessionId and the SessionTime, you can extract just about anything.

If you want to know even more, add a second **-v** to the command line to get a dump of every record in every block.

```
./bls -k -v -v /tmp/File002
bls: block.c:79 Dump block  80f8ad0: size=64512 BlkNum=1
               Hdrcksum=b1bdfd6d cksum=b1bdfd6d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=VOL_LABEL Strm=0 len=147 p=80f8b40
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=SOS_LABEL Strm=-7 len=122 p=80f8be7
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=1 Strm=UATTR len=86 p=80f8c75
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=2 Strm=UATTR len=90 p=80f8cdf
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=UATTR len=92 p=80f8d4d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=DATA len=54 p=80f8dbd
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=MD5 len=16 p=80f8e07
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=UATTR len=98 p=80f8e2b
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=DATA len=16 p=80f8ea1
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=MD5 len=16 p=80f8ec5
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=UATTR len=96 p=80f8ee9
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=DATA len=1783 p=80f8f5d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=MD5 len=16 p=80f9668
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=UATTR len=95 p=80f968c
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=80f96ff
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=8101713
bls: block.c:79 Dump block  80f8ad0: size=64512 BlkNum=2
               Hdrcksum=9acc1e7f cksum=9acc1e7f
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=contDATA len=4087 p=80f8b40
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=31970 p=80f9b4b
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=MD5 len=16 p=8101841
...
```

36.6 bextract

If you find yourself using **bextract**, you probably have done something wrong. For example, if you are trying to recover a file but are having problems, please see the Restoring When Things Go Wrong section of the Restore chapter of this manual.

Normally, you will restore files by running a **Restore** Job from the **Console** program. However, **bextract** can be used to extract a single file or a list of files from a Bacula tape or file. In fact, **bextract** can be a useful tool to restore files to an empty system assuming you are able to boot, you have statically linked **bextract** and you have an appropriate **bootstrap** file.

Please note that some of the current limitations of bextract are:

1. It cannot restore access control lists (ACL) that have been backed up along with the file data.
2. It cannot restore Win32 non-portable streams (typically default).
3. It cannot restore encrypted files.

4. The command line length is relatively limited, which means that you cannot enter a huge number of volumes. If you need to enter more volumes than the command line supports, please use a bootstrap file (see below).

It is called:

```
Usage: bextract [-d debug_level] <device-name> <directory-to-store-files>
       -b <file>       specify a bootstrap file
       -dnn            set debug level to nn
       -e <file>       exclude list
       -i <file>       include list
       -p              proceed inspite of I/O errors
       -V              specify Volume names (separated by |)
       -?              print this message
```

where **device-name** is the Archive Device (raw device name or full filename) of the device to be read, and **directory-to-store-files** is a path prefix to prepend to all the files restored.

NOTE: On Windows systems, if you specify a prefix of say d:/tmp, any file that would have been restored to **c:/My Documents** will be restored to **d:/tmp/My Documents**. That is, the original drive specification will be stripped. If no prefix is specified, the file will be restored to the original drive.

36.6.1 Extracting with Include or Exclude Lists

Using the **-e** option, you can specify a file containing a list of files to be excluded. Wildcards can be used in the exclusion list. This option will normally be used in conjunction with the **-i** option (see below). Both the **-e** and the **-i** options may be specified at the same time as the **-b** option. The bootstrap filters will be applied first, then the include list, then the exclude list.

Likewise, and probably more importantly, with the **-i** option, you can specify a file that contains a list (one file per line) of files and directories to include to be restored. The list must contain the full filename with the path. If you specify a path name only, all files and subdirectories of that path will be restored. If you specify a line containing only the filename (e.g. **myfile.txt**) it probably will not be extracted because you have not specified the full path.

For example, if the file **include-list** contains:

```
/home/kern/bacula
/usr/local/bin
```

Then the command:

```
./bextract -i include-list -V Volume /dev/nst0 /tmp
```

will restore from the Bacula archive **/dev/nst0** all files and directories in the backup from **/home/kern/bacula** and from **/usr/local/bin**. The restored files will be placed in a file of the original name under the directory **/tmp** (i.e. /tmp/home/kern/bacula/... and /tmp/usr/local/bin/...).

36.6.2 Extracting With a Bootstrap File

The **-b** option is used to specify a **bootstrap** file containing the information needed to restore precisely the files you want. Specifying a **bootstrap** file is optional but recommended because it gives you the most control over which files will be restored. For more details on the **bootstrap** file, please see Restoring Files with the Bootstrap File chapter of this document. Note, you may also use a bootstrap file produced by the **restore** command. For example:

```
./bextract -b bootstrap-file /dev/nst0 /tmp
```

The bootstrap file allows detailed specification of what files you want restored (extracted). You may specify a bootstrap file and include and/or exclude files at the same time. The bootstrap conditions will first be applied, and then each file record seen will be compared to the include and exclude lists.

36.6.3 Extracting From Multiple Volumes

If you wish to extract files that span several Volumes, you can specify the Volume names in the bootstrap file or you may specify the Volume names on the command line by separating them with a vertical bar. See the section above under the **bls** program entitled **Listing Multiple Volumes** for more information. The same techniques apply equally well to the **bextract** program or read the Bootstrap chapter of this document.

36.7 bscan

If you find yourself using this program, you have probably done something wrong. For example, the best way to recover a lost or damaged Bacula database is to reload the database by using the bootstrap file that was written when you saved it (default bacula-dir.conf file).

The **bscan** program can be used to re-create a database (catalog) records from the backup information written to one or more Volumes. This is normally needed only if one or more Volumes have been pruned or purged from your catalog so that the records on the Volume are no longer in the catalog, or for Volumes that you have archived.

With some care, it can also be used to synchronize your existing catalog with a Volume. Although we have never seen a case of bscan damaging a catalog, since bscan modifies your catalog, we recommend that you do a simple ASCII backup of your database before running **bscan** just to be sure. See Compacting Your Database for the details of making a copy of your database.

bscan can also be useful in a disaster recovery situation, after the loss of a hard disk, if you do not have a valid **bootstrap** file for reloading your system, or if a Volume has been recycled but not overwritten, you can use **bscan** to re-create your database, which can then be used to **restore** your system or a file to its previous state.

It is called:

```
Usage: bscan [options] <bacula-archive>
       -b bootstrap   specify a bootstrap file
       -c <file>      specify configuration file
       -d <nn>        set debug level to nn
       -m             update media info in database
       -n <name>      specify the database name (default bacula)
       -u <user>      specify database user name (default bacula)
       -P <password>  specify database password (default none)
       -h <host>      specify database host (default NULL)
       -p             proceed inspite of I/O errors
       -r             list records
       -s             synchronize or store in database
       -v             verbose
       -V <Volumes>   specify Volume names (separated by |)
       -w <dir>       specify working directory (default from conf file)
       -?             print this message
```

If you are using MySQL or PostgreSQL, there is no need to supply a working directory since in that case, bscan knows where the databases are. However, if you have provided security on your database, you may need to supply either the database name (**-b** option), the user name (**-u** option), and/or the password (**-p**) options.

NOTE: before **bscan** can work, it needs at least a bare bones valid database. If your database exists but some records are missing because they were pruned, then you are all set. If your database was lost or destroyed, then you must first ensure that you have the SQL program running (MySQL or PostgreSQL), then you must create the Bacula database (normally named bacula), and you must create the Bacula tables using the scripts in the **cats** directory. This is explained in the Installation chapter of the manual. Finally, before scanning into an empty database, you must start and stop the Director with the appropriate 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.

Forgetting for the moment the extra complications of a full rebuild of your catalog, let's suppose that you did a backup to Volumes "Vol001" and "Vol002", then sometime later all records of one or both those Volumes were pruned or purged from the database. By using **bscan** you can recreate the catalog entries for those Volumes and then use the **restore** command in the Console to restore whatever you want. A command something like:

```
bscan -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```

will give you an idea of what is going to happen without changing your catalog. Of course, you may need to change the path to the Storage daemon's conf file, the Volume name, and your tape (or disk) device name. This command must read the entire tape, so if it has a lot of data, it may take a long time, and thus you might want to immediately use the command listed below. Note, if you are writing to a disk file, replace the device name with the path to the directory that contains the Volumes. This must correspond to the Archive Device in the conf file.

Then to actually write or store the records in the catalog, add the **-s** option as follows:

```
 bscan -s -m -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```

When writing to the database, if bscan finds existing records, it will generally either update them if something is wrong or leave them alone. Thus if the

Volumes you are scanning are all or partially in the catalog already, no harm will be done to that existing data. Any missing data will simply be added.

If you have multiple tapes, you should scan them with:

```
bscan -s -m -c bacula-sd.conf -v -V Vol001\|Vol002\|Vol003 /dev/nst0
```

Since there is a limit on the command line length (511 bytes) accepted by **bscan**, if you have too many Volumes, you will need to manually create a bootstrap file. See the Bootstrap chapter of this manual for more details, in particular the section entitled Bootstrap for bscan.

You should, always try to specify the tapes in the order they are written. However, bscan can handle scanning tapes that are not sequential. Any incomplete records at the end of the tape will simply be ignored in that case. If you are simply repairing an existing catalog, this may be OK, but if you are creating a new catalog from scratch, it will leave your database in an incorrect state. If you do not specify all necessary Volumes on a single bscan command, bscan will not be able to correctly restore the records that span two volumes. In other words, it is much better to specify two or three volumes on a single bscan command rather than run bscan two or three times, each with a single volume.

Note, the restoration process using bscan is not identical to the original creation of the catalog data. This is because certain data such as Client records and other non-essential data such as volume reads, volume mounts, etc is not stored on the Volume, and thus is not restored by bscan. The results of bscanning are, however, perfectly valid, and will permit restoration of any or all the files in the catalog using the normal Bacula console commands. If you are starting with an empty catalog and expecting bscan to reconstruct it, you may be a bit disappointed, but at a minimum, you must ensure that your bacula-dir.conf file is the same as what it previously was – that is, it must contain all the appropriate Client resources so that they will be recreated in your new database **before** running bscan. Normally when the Director starts, it will recreate any missing Client records in the catalog. Another problem you will have is that even if the Volumes (Media records) are recreated in the database, they will not have their autochanger status and slots properly set. As a result, you will need to repair that by using the **update slots** command. There may be other considerations as well. Rather than bscanning, you should always attempt to recover you previous catalog backup.

36.7.1 Using bscan to Compare a Volume to an existing Catalog

If you wish to compare the contents of a Volume to an existing catalog without changing the catalog, you can safely do so if and only if you do **not** specify either the **-m** or the **-s** options. However, at this time (Bacula version 1.26), the comparison routines are not as good or as thorough as they should be, so we don't particularly recommend this mode other than for testing.

36.7.2 Using bscan to Recreate a Catalog from a Volume

This is the mode for which **bscan** is most useful. You can either **bscan** into a freshly created catalog, or directly into your existing catalog (after having made an ASCII copy as described above). Normally, you should start with a freshly created catalog that contains no data.

Starting with a single Volume named **TestVolume1**, you run a command such as:

```
./bscan -V TestVolume1 -v -s -m -c bacula-sd.conf /dev/nst0
```

If there is more than one volume, simply append it to the first one separating it with a vertical bar. You may need to precede the vertical bar with a forward slash escape the shell – e.g. **TestVolume1\—TestVolume2**. The **-v** option was added for verbose output (this can be omitted if desired). The **-s** option that tells **bscan** to store information in the database. The physical device name **/dev/nst0** is specified after all the options.

For example, after having done a full backup of a directory, then two incrementals, I reinitialized the SQLite database as described above, and using the bootstrap.bsr file noted above, I entered the following command:

```
./bscan -b bootstrap.bsr -v -s -c bacula-sd.conf /dev/nst0
```

which produced the following output:

```
bscan: bscan.c:182 Using Database: bacula, User: bacula
bscan: bscan.c:673 Created Pool record for Pool: Default
bscan: bscan.c:271 Pool type "Backup" is OK.
bscan: bscan.c:632 Created Media record for Volume: TestVolume1
```

```
bscan: bscan.c:298 Media type "DDS-4" is OK.
bscan: bscan.c:307 VOL_LABEL: OK for Volume: TestVolume1
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=1 record for original JobId=2
bscan: bscan.c:717 Created FileSet record "Kerns Files"
bscan: bscan.c:819 Updated Job termination record for new JobId=1
bscan: bscan.c:905 Created JobMedia record JobId 1, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=2 record for original JobId=3
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=2
bscan: bscan.c:905 Created JobMedia record JobId 2, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=3 record for original JobId=4
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=3
bscan: bscan.c:905 Created JobMedia record JobId 3, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:652 Updated Media record at end of Volume: TestVolume1
bscan: bscan.c:428 End of Volume. VolFiles=3 VolBlocks=57 VolBytes=10,027,437
```

The key points to note are that **bscan** prints a line when each major record is created. Due to the volume of output, it does not print a line for each file record unless you supply the **-v** option twice or more on the command line.

In the case of a Job record, the new JobId will not normally be the same as the original Jobid. For example, for the first JobId above, the new JobId is 1, but the original JobId is 2. This is nothing to be concerned about as it is the normal nature of databases. **bscan** will keep everything straight.

Although **bscan** claims that it created a Client record for Client: Rufus three times, it was actually only created the first time. This is normal.

You will also notice that it read an end of file after each Job (Got EOF on device ...). Finally the last line gives the total statistics for the bscan.

If you had added a second **-v** option to the command line, Bacula would have been even more verbose, dumping virtually all the details of each Job record it encountered.

Now if you start Bacula and enter a **list jobs** command to the console program, you will get:

```
+-------+----------+------------------+------+-----+----------+----------+---------+
| JobId | Name     | StartTime        | Type | Lvl | JobFiles | JobBytes | JobStat |
+-------+----------+------------------+------+-----+----------+----------+---------+
| 1     | kernsave | 2002-10-07 14:59 | B    | F   | 84       | 4180207  | T       |
```

```
| 2     | kernsave | 2002-10-07 15:00 | B    | I   | 15       | 2170314  | T       |
| 3     | kernsave | 2002-10-07 15:01 | B    | I   | 33       | 3662184  | T       |
+-------+----------+------------------+------+-----+----------+----------+---------+
```

which corresponds virtually identically with what the database contained before it was re-initialized and restored with bscan. All the Jobs and Files found on the tape are restored including most of the Media record. The Volume (Media) records restored will be marked as **Full** so that they cannot be rewritten without operator intervention.

It should be noted that **bscan** cannot restore a database to the exact condition it was in previously because a lot of the less important information contained in the database is not saved to the tape. Nevertheless, the reconstruction is sufficiently complete, that you can run **restore** against it and get valid results.

An interesting aspect of restoring a catalog backup using **bscan** is that the backup was made while Bacula was running and writing to a tape. At the point the backup of the catalog is made, the tape Bacula is writing to will have say 10 files on it, but after the catalog backup is made, there will be 11 files on the tape Bacula is writing. This there is a difference between what is contained in the backed up catalog and what is actually on the tape. If after restoring a catalog, you attempt to write on the same tape that was used to backup the catalog, Bacula will detect the difference in the number of files registered in the catalog compared to what is on the tape, and will mark the tape in error.

There are two solutions to this problem. The first is possibly the simplest and is to mark the volume as Used before doing any backups. The second is to manually correct the number of files listed in the Media record of the catalog. This procedure is documented elsewhere in the manual and involves using the **update volume** command in **bconsole**.

36.7.3 Using bscan to Correct the Volume File Count

If the Storage daemon crashes during a backup Job, the catalog will not be properly updated for the Volume being used at the time of the crash. This means that the Storage daemon will have written say 20 files on the tape, but the catalog record for the Volume indicates only 19 files.

Bacula refuses to write on a tape that contains a different number of files from what is in the catalog. To correct this situation, you may run a **bscan** with the **-m** option (but **without** the **-s** option) to update only the final Media record for the Volumes read.

36.7.4 After bscan

If you use **bscan** to enter the contents of the Volume into an existing catalog, you should be aware that the records you entered may be immediately pruned during the next job, particularly if the Volume is very old or had been previously purged. To avoid this, after running **bscan**, you can manually set the volume status (VolStatus) to **Read-Only** by using the **update** command in the catalog. This will allow you to restore from the volume without having it immediately purged. When you have restored and backed up the data, you can reset the VolStatus to **Used** and the Volume will be purged from the catalog.

36.8 bcopy

The **bcopy** program can be used to copy one **Bacula** archive file to another. For example, you may copy a tape to a file, a file to a tape, a file to a file, or a tape to a tape. For tape to tape, you will need two tape drives. (a later version is planned that will buffer it to disk). In the process of making the copy, no record of the information written to the new Volume is stored in the catalog. This means that the new Volume, though it contains valid backup data, cannot be accessed directly from existing catalog entries. If you wish to be able to use the Volume with the Console restore command, for example, you must first bscan the new Volume into the catalog.

36.8.1 bcopy Command Options

```
Usage: bcopy [-d debug_level] <input-archive> <output-archive>
       -b bootstrap      specify a bootstrap file
       -c <file>         specify configuration file
       -dnn              set debug level to nn
       -i                specify input Volume names (separated by |)
       -o                specify output Volume names (separated by |)
       -p                proceed inspite of I/O errors
       -v                verbose
       -w dir            specify working directory (default /tmp)
       -?                print this message
```

By using a **bootstrap** file, you can copy parts of a Bacula archive file to another archive.

One of the objectives of this program is to be able to recover as much data as possible from a damaged tape. However, the current version does not yet have this feature.

As this is a new program, any feedback on its use would be appreciated. In addition, I only have a single tape drive, so I have never been able to test this program with two tape drives.

36.9 btape

This program permits a number of elementary tape operations via a tty command interface. It works only with tapes and not with other kinds of Bacula storage media (DVD, File, ...). The **test** command, described below, can be very useful for testing older tape drive compatibility problems. Aside from initial testing of tape drive compatibility with **Bacula**, **btape** will be mostly used by developers writing new tape drivers.

btape can be dangerous to use with existing **Bacula** tapes because it will relabel a tape or write on the tape if so requested regardless that the tape may contain valuable data, so please be careful and use it only on blank tapes.

To work properly, **btape** needs to read the Storage daemon's configuration file. As a default, it will look for **bacula-sd.conf** in the current directory. If your configuration file is elsewhere, please use the **-c** option to specify where.

The physical device name must be specified on the command line, and this same device name must be present in the Storage daemon's configuration file read by **btape**

```
Usage: btape <options> <device_name>
       -b <file>   specify bootstrap file
       -c <file>   set configuration file to file
       -d <nn>     set debug level to nn
       -p          proceed inspite of I/O errors
       -s          turn off signals
       -v          be verbose
       -?          print this message.
```

36.9.1 Using btape to Verify your Tape Drive

An important reason for this program is to ensure that a Storage daemon configuration file is defined so that Bacula will correctly read and write tapes.

It is highly recommended that you run the **test** command before running your first Bacula job to ensure that the parameters you have defined for your storage device (tape drive) will permit **Bacula** to function properly. You only need to mount a blank tape, enter the command, and the output should be reasonably self explanatory. Please see the Tape Testing Chapter of this manual for the details.

36.9.2 btape Commands

The full list of commands are:

```
Command    Description
=======    ===========
autochanger test autochanger
bsf        backspace file
bsr        backspace record
cap        list device capabilities
clear      clear tape errors
eod        go to end of Bacula data for append
eom        go to the physical end of medium
fill       fill tape, write onto second volume
unfill     read filled tape
fsf        forward space a file
fsr        forward space a record
help       print this command
label      write a Bacula label to the tape
load       load a tape
quit       quit btape
rawfill    use write() to fill tape
readlabel  read and print the Bacula tape label
rectest    test record handling functions
rewind     rewind the tape
scan       read() tape block by block to EOT and report
scanblocks Bacula read block by block to EOT and report
status     print tape status
test       General test Bacula tape functions
weof       write an EOF on the tape
wr         write a single Bacula block
rr         read a single record
qfill      quick fill command
```

The most useful commands are:

- test – test writing records and EOF marks and reading them back.
- fill – completely fill a volume with records, then write a few records on a second volume, and finally, both volumes will be read back. This

command writes blocks containing random data, so your drive will not be able to compress the data, and thus it is a good test of the real physical capacity of your tapes.

- readlabel – read and dump the label on a Bacula tape.
- cap – list the device capabilities as defined in the configuration file and as perceived by the Storage daemon.

The **readlabel** command can be used to display the details of a Bacula tape label. This can be useful if the physical tape label was lost or damaged.

In the event that you want to relabel a **Bacula**, you can simply use the **label** command which will write over any existing label. However, please note for labeling tapes, we recommend that you use the **label** command in the **Console** program since it will never overwrite a valid Bacula tape.

36.10 Other Programs

The following programs are general utility programs and in general do not need a configuration file nor a device name.

36.11 bsmtp

bsmtp is a simple mail transport program that permits more flexibility than the standard mail programs typically found on Unix systems. It can even be used on Windows machines.

It is called:

```
Usage: bsmtp [-f from] [-h mailhost] [-s subject] [-c copy] [recipient ...]
       -c          set the Cc: field
       -dnn        set debug level to nn
       -f          set the From: field
       -h          use mailhost:port as the bsmtp server
       -l          limit the lines accepted to nn
       -s          set the Subject: field
       -?          print this message.
```

If the **-f** option is not specified, **bsmtp** will use your userid. If the option **-h** is not specified **bsmtp** will use the value in the environment variable **bsmtpSERVER** or if there is none **localhost**. By default port 25 is used.

If a line count limit is set with the **-l** option, **bsmtp** will not send an email with a body text exceeding that number of lines. This is especially useful for large restore job reports where the list of files restored might produce very long mails your mail-server would refuse or crash. However, be aware that you will probably suppress the job report and any error messages unless you check the log file written by the Director (see the messages resource in this manual for details).

recipients is a space separated list of email recipients.

The body of the email message is read from standard input.

An example of the use of **bsmtp** would be to put the following statement in the **Messages** resource of your **bacula-dir.conf** file. Note, these commands should appear on a single line each.

```
mailcommand = "/home/bacula/bin/bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
              -s \"Bacula: %t %e of %c %l\" %r"
operatorcommand = "/home/bacula/bin/bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
                  -s \"Bacula: Intervention needed for %j\" %r"
```

Where you replace **/home/bacula/bin** with the path to your **Bacula** binary directory, and you replace **mail.domain.com** with the fully qualified name of your bsmtp (email) server, which normally listens on port 25. For more details on the substitution characters (e.g. %r) used in the above line, please see the documentation of the MailCommand in the Messages Resource chapter of this manual.

It is HIGHLY recommended that you test one or two cases by hand to make sure that the **mailhost** that you specified is correct and that it will accept your email requests. Since **bsmtp** always uses a TCP connection rather than writing in the spool file, you may find that your **from** address is being rejected because it does not contain a valid domain, or because your message is caught in your spam filtering rules. Generally, you should specify a fully qualified domain name in the **from** field, and depending on whether your bsmtp gateway is Exim or Sendmail, you may need to modify the syntax of the from part of the message. Please test.

When running **bsmtp** by hand, you will need to terminate the message by entering a ctl-d in column 1 of the last line.

If you are getting incorrect dates (e.g. 1970) and you are running with a non-English language setting, you might try adding a LANG="en_US" immediately before the bsmtp call.

36.12 dbcheck

dbcheck is a simple program that will search for logical inconsistencies in the Bacula tables in your database, and optionally fix them. It is a database maintenance routine, in the sense that it can detect and remove unused rows, but it is not a database repair routine. To repair a database, see the tools furnished by the database vendor. Normally dbcheck should never need to be run, but if Bacula has crashed or you have a lot of Clients, Pools, or Jobs that you have removed, it could be useful.

The **dbcheck** program can be found in the **<bacula-source>/src/tools** directory of the source distribution. Though it is built with the make process, it is not normally "installed".

It is called:

```
Usage: dbcheck [-c config] [-C catalog name] [-d debug_level]
<working-directory> <bacula-database> <user> <password> [<dbhost>]
       -b              batch mode
       -C              catalog name in the director conf file
       -c              director conf filename
       -dnn            set debug level to nn
       -f              fix inconsistencies
       -v              verbose
       -?              print this message
```

If the **-c** option is given with the Director's conf file, there is no need to enter any of the command line arguments, in particular the working directory as dbcheck will read them from the file.

If the **-f** option is specified, **dbcheck** will repair (**fix**) the inconsistencies it finds. Otherwise, it will report only.

If the **-b** option is specified, **dbcheck** will run in batch mode, and it will proceed to examine and fix (if -f is set) all programmed inconsistency checks. If the **-b** option is not specified, **dbcheck** will enter interactive mode and prompt with the following:

```
Hello, this is the database check/correct program.
Please select the function you want to perform.
     1) Toggle modify database flag
     2) Toggle verbose flag
     3) Repair bad Filename records
     4) Repair bad Path records
     5) Eliminate duplicate Filename records
     6) Eliminate duplicate Path records
```

```
     7) Eliminate orphaned Jobmedia records
     8) Eliminate orphaned File records
     9) Eliminate orphaned Path records
    10) Eliminate orphaned Filename records
    11) Eliminate orphaned FileSet records
    12) Eliminate orphaned Client records
    13) Eliminate orphaned Job records
    14) Eliminate all Admin records
    15) Eliminate all Restore records
    16) All (3-15)
    17) Quit
Select function number:
```

By entering 1 or 2, you can toggle the modify database flag (-f option) and the verbose flag (-v). It can be helpful and reassuring to turn off the modify database flag, then select one or more of the consistency checks (items 3 through 9) to see what will be done, then toggle the modify flag on and re-run the check.

The inconsistencies examined are the following:

- Duplicate filename records. This can happen if you accidentally run two copies of Bacula at the same time, and they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. If this is the case, you will receive error messages during Jobs warning of duplicate database records. If you are not getting these error messages, there is no reason to run this check.
- Repair bad Filename records. This checks and corrects filenames that have a trailing slash. They should not.
- Repair bad Path records. This checks and corrects path names that do not have a trailing slash. They should.
- Duplicate path records. This can happen if you accidentally run two copies of Bacula at the same time, and they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. See the item above for why this occurs and how you know it is happening.
- Orphaned JobMedia records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding JobMedia record (one for each Volume used in the Job) was not deleted. Normally, this should not happen, and even if it does, these records generally do not take much space in your database. However, by running this check, you can eliminate any such orphans.

- Orphaned File records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding File record (one for each Volume used in the Job) was not deleted. Note, searching for these records can be **very** time consuming (i.e. it may take hours) for a large database. Normally this should not happen as Bacula takes care to prevent it. Just the same, this check can remove any orphaned File records. It is recommended that you run this once a year since orphaned File records can take a large amount of space in your database. You might want to ensure that you have indexes on JobId, FilenameId, and PathId for the File table in your catalog before running this command.

- Orphaned Path records. This condition happens any time a directory is deleted from your system and all associated Job records have been purged. During standard purging (or pruning) of Job records, Bacula does not check for orphaned Path records. As a consequence, over a period of time, old unused Path records will tend to accumulate and use space in your database. This check will eliminate them. It is recommended that you run this check at least once a year.

- Orphaned Filename records. This condition happens any time a file is deleted from your system and all associated Job records have been purged. This can happen quite frequently as there are quite a large number of files that are created and then deleted. In addition, if you do a system update or delete an entire directory, there can be a very large number of Filename records that remain in the catalog but are no longer used.

  During standard purging (or pruning) of Job records, Bacula does not check for orphaned Filename records. As a consequence, over a period of time, old unused Filename records will accumulate and use space in your database. This check will eliminate them. It is strongly recommended that you run this check at least once a year, and for large database (more than 200 Megabytes), it is probably better to run this once every 6 months.

- Orphaned Client records. These records can remain in the database long after you have removed a client.

- Orphaned Job records. If no client is defined for a job or you do not run a job for a long time, you can accumulate old job records. This option allow you to remove jobs that are not attached to any client (and thus useless).

- All Admin records. This command will remove all Admin records, regardless of their age.

- All Restore records. This command will remove all Restore records, regardless of their age.

By the way, I personally run dbcheck only where I have messed up my database due to a bug in developing Bacula code, so normally you should never need to run dbcheck in spite of the recommendations given above, which are given so that users don't waste their time running dbcheck too often.

36.13 bregex

bregex is a simple program that will allow you to test regular expressions against a file of data. This can be useful because the regex libraries on most systems differ, and in addition, regex expressions can be complicated.

bregex is found in the src/tools directory and it is normally installed with your system binaries. To run it, use:

```
Usage: bregex [-d debug_level] -f <data-file>
       -f          specify file of data to be matched
       -l          suppress line numbers
       -n          print lines that do not match
       -?          print this message.
```

The <data-file> is a filename that contains lines of data to be matched (or not) against one or more patterns. When the program is run, it will prompt you for a regular expression pattern, then apply it one line at a time against the data in the file. Each line that matches will be printed preceded by its line number. You will then be prompted again for another pattern.

Enter an empty line for a pattern to terminate the program. You can print only lines that do not match by using the -n option, and you can suppress printing of line numbers with the -l option.

This program can be useful for testing regex expressions to be applied against a list of filenames.

36.14 bwild

bwild is a simple program that will allow you to test wild-card expressions against a file of data.

bwild is found in the src/tools directory and it is normally installed with your system binaries. To run it, use:

```
Usage: bwild [-d debug_level] -f <data-file>
       -f          specify file of data to be matched
       -l          suppress line numbers
       -n          print lines that do not match
       -?          print this message.
```

The <data-file> is a filename that contains lines of data to be matched (or not) against one or more patterns. When the program is run, it will prompt you for a wild-card pattern, then apply it one line at a time against the data in the file. Each line that matches will be printed preceded by its line number. You will then be prompted again for another pattern.

Enter an empty line for a pattern to terminate the program. You can print only lines that do not match by using the -n option, and you can suppress printing of line numbers with the -l option.

This program can be useful for testing wild expressions to be applied against a list of filenames.

36.15 testfind

testfind permits listing of files using the same search engine that is used for the **Include** resource in Job resources. Note, much of the functionality of this program (listing of files to be included) is present in the estimate command in the Console program.

The original use of testfind was to ensure that Bacula's file search engine was correct and to print some statistics on file name and path length. However, you may find it useful to see what bacula would do with a given **Include** resource. The **testfind** program can be found in the **<bacula-source>/src/tools** directory of the source distribution. Though it is built with the make process, it is not normally "installed".

It is called:

```
Usage: testfind [-d debug_level] [-] [pattern1 ...]
       -a          print extended attributes (Win32 debug)
```

```
       -dnn        set debug level to nn
       -           read pattern(s) from stdin
       -?          print this message.
Patterns are used for file inclusion -- normally directories.
Debug level>= 1 prints each file found.
Debug level>= 10 prints path/file for catalog.
Errors are always printed.
Files/paths truncated is a number with len> 255.
Truncation is only in the catalog.
```

Where a pattern is any filename specification that is valid within an **Include** resource definition. If none is specified, / (the root directory) is assumed. For example:

```
./testfind /bin
```

Would print the following:

```
Dir: /bin
Reg: /bin/bash
Lnk: /bin/bash2 -> bash
Lnk: /bin/sh -> bash
Reg: /bin/cpio
Reg: /bin/ed
Lnk: /bin/red -> ed
Reg: /bin/chgrp
...
Reg: /bin/ipcalc
Reg: /bin/usleep
Reg: /bin/aumix-minimal
Reg: /bin/mt
Lnka: /bin/gawk-3.1.0 -> /bin/gawk
Reg: /bin/pgawk
Total files    : 85
Max file length: 13
Max path length: 5
Files truncated: 0
Paths truncated: 0
```

Even though **testfind** uses the same search engine as **Bacula**, each directory to be listed, must be entered as a separate command line entry or entered one line at a time to standard input if the **-** option was specified.

Specifying a debug level of one (i.e. **-d1**) on the command line will cause **testfind** to print the raw filenames without showing the Bacula internal file type, or the link (if any). Debug levels of 10 or greater cause the filename and the path to be separated using the same algorithm that is used when putting filenames into the Catalog database.

Chapter 37

Testing Your Tape Drive With Bacula

This chapter is concerned with testing and configuring your tape drive to make sure that it will work properly with Bacula using the **btape** program.

37.1 Get Your Tape Drive Working

In general, you should follow the following steps to get your tape drive to work with Bacula. Start with a tape mounted in your drive. If you have an autochanger, load a tape into the drive. We use **/dev/nst0** as the tape drive name, you will need to adapt it according to your system.

Do not proceed to the next item until you have succeeded with the previous one.

1. Make sure that Bacula (the Storage daemon) is not running or that you have **unmount**ed the drive you will use for testing.

2. Use tar to write to, then read from your drive:

```
mt -f /dev/nst0 rewind
tar cvf /dev/nst0 .
mt -f /dev/nst0 rewind
tar tvf /dev/nst0
```

3. Make sure you have a valid and correct Device resource corresponding to your drive. For Linux users, generally, the default one works. For FreeBSD users, there are two possible Device configurations (see below). For other drives and/or OSes, you will need to first ensure that your system tape modes are properly setup (see below), then possibly modify you Device resource depending on the output from the btape program (next item). When doing this, you should consult the Storage Daemon Configuration of this manual.

4. If you are using a Fibre Channel to connect your tape drive to Bacula, please be sure to disable any caching in the NSR (network storage router, which is a Fibre Channel to SCSI converter).

5. Run the btape **test** command:

```
./btape -c bacula-sd.conf /dev/nst0
test
```

   It isn't necessary to run the autochanger part of the test at this time, but do not go past this point until the basic test succeeds. If you do have an autochanger, please be sure to read the Autochanger chapter of this manual.

6. Run the btape **fill** command, preferably with two volumes. This can take a long time. If you have an autochanger and it is configured, Bacula will automatically use it. If you do not have it configured, you can manually issue the appropriate **mtx** command, or press the autochanger buttons to change the tape when requested to do so.

7. FreeBSD users, if you have a pre-5.0 system run the **tapetest** program, and make sure your system is patched if necessary. The tapetest program can be found in the platform/freebsd directory. The instructions for its use are at the top of the file.

8. Run Bacula, and backup a reasonably small directory, say 60 Megabytes. Do three successive backups of this directory.

9. Stop Bacula, then restart it. Do another full backup of the same directory. Then stop and restart Bacula.

10. Do a restore of the directory backed up, by entering the following restore command, being careful to restore it to an alternate location:

```
restore select all done
yes
```

Do a **diff** on the restored directory to ensure it is identical to the original directory. If you are going to backup multiple different systems (Linux, Windows, Mac, Solaris, FreeBSD, ...), be sure you test the restore on each system type.

11. If you have an autochanger, you should now go back to the btape program and run the autochanger test:

    ```
    ./btape -c bacula-sd.conf /dev/nst0
    auto
    ```

    Adjust your autochanger as necessary to ensure that it works correctly. See the Autochanger chapter of this manual for a complete discussion of testing your autochanger.

12. We strongly recommend that you use a dedicated SCSI controller for your tape drives. Scanners are known to induce serious problems with the SCSI bus, causing it to reset. If the SCSI bus is reset while Bacula has the tape drive open, it will most likely be fatal to your tape since the drive will rewind. These kinds of problems show up in the system log. For example, the following was most likely caused by a scanner:

    ```
    Feb 14 17:29:55 epohost kernel: (scsi0:A:2:0): No or incomplete CDB sent to device.
    Feb 14 17:29:55 epohost kernel: scsi0: Issued Channel A Bus Reset. 1 SCBs aborted
    ```

If you have reached this point, you stand a good chance of having everything work. If you get into trouble at any point, **carefully** read the documentation given below. If you cannot get past some point, ask the **bacula-users** email list, but specify which of the steps you have successfully completed. In particular, you may want to look at the Tips for Resolving Problems section below.

37.1.1 Problems When no Tape in Drive

When Bacula was first written the Linux 2.4 kernel permitted opening the drive whether or not there was a tape in the drive. Thus the Bacula code is based on the concept that if the drive cannot be opened, there is a serious problem, and the job is failed.

With version 2.6 of the Linux kernel, if there is no tape in the drive, the OS will wait two minutes (default) and then return a failure, and consequently, Bacula version 1.36 and below will fail the job. This is important to keep in mind, because if you use an option such as **Offline on Unmount = yes**, there will be a point when there is no tape in the drive, and if another job starts or if Bacula asks the operator to mount a tape, when Bacula attempts to open the drive (about a 20 minute delay), it will fail and Bacula will fail the job.

In version 1.38.x, the Bacula code partially gets around this problem – at least in the initial open of the drive. However, functions like Polling the drive do not work correctly if there is no tape in the drive. Providing you do not use **Offline on Unmount = yes**, you should not experience job failures as mentioned above. If you do experience such failures, you can also increase the **Maximum Open Wait** time interval, which will give you more time to mount the next tape before the job is failed.

37.1.2 Specifying the Configuration File

Starting with version 1.27, each of the tape utility programs including the **btape** program requires a valid Storage daemon configuration file (actually, the only part of the configuration file that **btape** needs is the **Device** resource definitions). This permits **btape** to find the configuration parameters for your archive device (generally a tape drive). Without those parameters, the testing and utility programs do not know how to properly read and write your drive. By default, they use **bacula-sd.conf** in the current directory, but you may specify a different configuration file using the **-c** option.

37.1.3 Specifying a Device Name For a Tape

btape device-name where the Volume can be found. In the case of a tape, this is the physical device name such as **/dev/nst0** or **/dev/rmt/0ubn** depending on your system that you specify on the Archive Device directive. For the program to work, it must find the identical name in the Device resource of the configuration file. If the name is not found in the list of physical names, the utility program will compare the name you entered to the Device names (rather than the Archive device names).

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.

See below for specifying Volume names.

37.1.4 Specifying a Device Name For a File

If you are attempting to read or write an archive file rather than a tape, the **device-name** should be the full path to the archive location including the filename. The filename (last part of the specification) will be stripped and used as the Volume name, and the path (first part before the filename) must have the same entry in the configuration file. So, the path is equivalent to the archive device name, and the filename is equivalent to the volume name.

37.2 btape

This program permits a number of elementary tape operations via a tty command interface. The **test** command, described below, can be very useful for testing tape drive compatibility problems. Aside from initial testing of tape drive compatibility with **Bacula**, **btape** will be mostly used by developers writing new tape drivers.

btape can be dangerous to use with existing **Bacula** tapes because it will relabel a tape or write on the tape if so requested regardless of whether or not the tape contains valuable data, so please be careful and use it only on blank tapes.

To work properly, **btape** needs to read the Storage daemon's configuration file. As a default, it will look for **bacula-sd.conf** in the current directory. If your configuration file is elsewhere, please use the **-c** option to specify where.

The physical device name or the Device resource name must be specified on the command line, and this same device name must be present in the Storage daemon's configuration file read by **btape**

```
Usage: btape [options] device_name
       -b <file>   specify bootstrap file
       -c <file>   set configuration file to file
       -d <nn>     set debug level to nn
       -p          proceed inspite of I/O errors
       -s          turn off signals
       -v          be verbose
       -?          print this message.
```

37.2.1 Using btape to Verify your Tape Drive

An important reason for this program is to ensure that a Storage daemon configuration file is defined so that Bacula will correctly read and write tapes.

It is highly recommended that you run the **test** command before running your first Bacula job to ensure that the parameters you have defined for your storage device (tape drive) will permit **Bacula** to function properly. You only need to mount a blank tape, enter the command, and the output should be reasonably self explanatory. For example:

```
(ensure that Bacula is not running)
./btape -c /usr/bin/bacula/bacula-sd.conf /dev/nst0
```

The output will be:

```
Tape block granularity is 1024 bytes.
btape: btape.c:376 Using device: /dev/nst0
*
```

Enter the test command:

```
test
```

The output produced should be something similar to the following: I've cut the listing short because it is frequently updated to have new tests.

```
=== Append files test ===
This test is essential to Bacula.
I'm going to write one record  in file 0,
                   two records in file 1,
             and three records in file 2
btape: btape.c:387 Rewound /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
```

```
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:387 Rewound /dev/nst0
btape: btape.c:693 Now moving to end of media.
btape: btape.c:427 Moved to end of media
We should be in file 3. I am at file 3. This is correct!
Now the important part, I am going to attempt to append to the tape.
...
=== End Append files test ===
```

If you do not successfully complete the above test, please resolve the problem(s) before attempting to use **Bacula**. Depending on your tape drive, the test may recommend that you add certain records to your configuration. We strongly recommend that you do so and then re-run the above test to insure it works the first time.

Some of the suggestions it provides for resolving the problems may or may not be useful. If at all possible avoid using fixed blocking. If the test suddenly starts to print a long series of:

```
Got EOF on tape.
Got EOF on tape.
...
```

then almost certainly, you are running your drive in fixed block mode rather than variable block mode. See below for more help of resolving fix versus variable block problems.

It is also possible that you have your drive set in SysV tape drive mode. The drive must use BSD tape conventions. See the section above on setting your **Archive device** correctly.

For FreeBSD users, please see the notes below for doing further testing of your tape drive.

37.2.2 Linux SCSI Tricks

You can find out what SCSI devices you have by doing:

```
lsscsi
```

Typical output is:

```
[0:0:0:0]    disk    ATA      ST3160812AS      3.AD  /dev/sda
[2:0:4:0]    tape    HP       Ultrium 2-SCSI   F6CH  /dev/st0
[2:0:5:0]    tape    HP       Ultrium 2-SCSI   F6CH  /dev/st1
[2:0:6:0]    mediumx OVERLAND LXB              0107  -
[2:0:9:0]    tape    HP       Ultrium 1-SCSI   E50H  /dev/st2
[2:0:10:0]   mediumx OVERLAND LXB              0107  -
```

There are two drives in one autochanger: /dev/st0 and /dev/st1 and a third tape drive at /dev/st2. For using them with Bacula, one would normally reference them as /dev/nst0 ... /dev/nst2. Not also, there are two different autochangers identified as "mediumx OVERLAND LXB". They can be addressed via their /dev/sgN designation, which can be obtained by counting from the beginning as 0 to each changer. In the above case, the two changers are located on /dev/sg3 and /dev/sg5. The one at /dev/sg3, controls drives /dev/nst0 and /dev/nst1; and the one at /dev/sg5 controles drive /dev/nst2.

If you do not have the **lsscsi** command, you can obtain the same information as follows:

```
cat /proc/scsi/scsi
```

For the above example with the three drives and two autochangers, I get:

```
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: ST3160812AS      Rev: 3.AD
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: HP       Model: Ultrium 2-SCSI   Rev: F6CH
  Type:   Sequential-Access                ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 05 Lun: 00
  Vendor: HP       Model: Ultrium 2-SCSI   Rev: F6CH
  Type:   Sequential-Access                ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 06 Lun: 00
  Vendor: OVERLAND Model: LXB              Rev: 0107
  Type:   Medium Changer                   ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 09 Lun: 00
  Vendor: HP       Model: Ultrium 1-SCSI   Rev: E50H
  Type:   Sequential-Access                ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 10 Lun: 00
  Vendor: OVERLAND Model: LXB              Rev: 0107
  Type:   Medium Changer                   ANSI SCSI revision: 02
```

As an additional example, I get the following (on a different machine from the above example):

```
Attached devices:
Host: scsi2 Channel: 00 Id: 01 Lun: 00
  Vendor: HP       Model: C5713A           Rev: H107
  Type:   Sequential-Access                ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: SONY     Model: SDT-10000        Rev: 0110
  Type:   Sequential-Access                ANSI SCSI revision: 02
```

The above represents first an autochanger and second a simple tape drive. The HP changer (the first entry) uses the same SCSI channel for data and for control, so in Bacula, you would use:

```
Archive Device = /dev/nst0
Changer Device = /dev/sg0
```

If you want to remove the SDT-10000 device, you can do so as root with:

```
echo "scsi remove-single-device 2 0 4 0">/proc/scsi/scsi
```

and you can put add it back with:

```
echo "scsi add-single-device 2 0 4 0">/proc/scsi/scsi
```

where the 2 0 4 0 are the Host, Channel, Id, and Lun as seen on the output from **cat /proc/scsi/scsi**. Note, the Channel must be specified as numeric.

Below is a slightly more complicated output, which is a single autochanger with two drives, and which operates the changer on a different channel from from the drives:

```
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: WDC WD1600JD-75H Rev: 08.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: HP       Model: Ultrium 2-SCSI   Rev: F6CH
  Type:   Sequential-Access                ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 05 Lun: 00
  Vendor: HP       Model: Ultrium 2-SCSI   Rev: F6CH
  Type:   Sequential-Access                ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 06 Lun: 00
  Vendor: OVERLAND Model: LXB              Rev: 0106
  Type:   Medium Changer                   ANSI SCSI revision: 02
```

The above tape drives are accessed on /dev/nst0 and /dev/nst1, while the control channel for those two drives is /dev/sg3.

37.3 Tips for Resolving Problems

37.3.1 Bacula Saves But Cannot Restore Files

If you are getting error messages such as:

```
Volume data error at 0:1! Wanted block-id: "BB02", got "". Buffer discarded
```

It is very likely that Bacula has tried to do block positioning and ended up at an invalid block. This can happen if your tape drive is in fixed block mode while Bacula's default is variable blocks. Note that in such cases, Bacula is perfectly able to write to your Volumes (tapes), but cannot position to read them.

There are two possible solutions.

1. The first and best is to always ensure that your drive is in variable block mode. Note, it can switch back to fixed block mode on a reboot or if another program uses the drive. So on such systems you need to modify the Bacula startup files to explicitly set:

   ```
   mt -f /dev/nst0 defblksize 0
   ```

   or whatever is appropriate on your system. Note, if you are running a Linux system, and the above command does not work, it is most likely because you have not loaded the appropriate **mt** package, which is often called **mt_st**, but may differ according to your distribution.

2. The second possibility, especially, if Bacula wrote while the drive was in fixed block mode, is to turn off block positioning in Bacula. This is done by adding:

   ```
   Block Positioning = no
   ```

   to the Device resource. This is not the recommended procedure because it can enormously slow down recovery of files, but it may help where all else fails. This directive is available in version 1.35.5 or later (and not yet tested).

If you are getting error messages such as:

```
Volume data error at 0:0!
Block checksum mismatch in block=0 len=32625 calc=345678 blk=123456
```

You are getting tape read errors, and this is most likely due to one of the following things:

1. An old or bad tape.
2. A dirty drive that needs cleaning (particularly for DDS drives).
3. A loose SCSI cable.
4. Old firmware in your drive. Make sure you have the latest firmware loaded.
5. Computer memory errors.
6. Over-clocking your CPU.
7. A bad SCSI card.

37.3.2 Bacula Cannot Open the Device

If you get an error message such as:

```
dev open failed: dev.c:265 stored: unable to open
device /dev/nst0:> ERR=No such device or address
```

the first time you run a job, it is most likely due to the fact that you specified the incorrect device name on your **Archive Device**.

If Bacula works fine with your drive, then all off a sudden you get error messages similar to the one shown above, it is quite possible that your driver module is being removed because the kernel deems it idle. This is done via **crontab** with the use of **rmmod -a**. To fix the problem, you can remove this entry from **crontab**, or you can manually **modprob** your driver module (or add it to the local startup script). Thanks to Alan Brown for this tip.

37.3.3 Incorrect File Number

When Bacula moves to the end of the medium, it normally uses the **ioctl(MTEOM)** function. Then Bacula uses the **ioctl(MTIOCGET)** function to retrieve the current file position from the **mt_fileno** field. Some SCSI tape drivers will use a fast means of seeking to the end of the medium and in doing so, they will not know the current file position and hence return a **-1**. As a consequence, if you get **"This is NOT correct!"** in the positioning tests, this may be the cause. You must correct this condition in order for Bacula to work.

There are two possible solutions to the above problem of incorrect file number:

- Figure out how to configure your SCSI driver to keep track of the file position during the MTEOM request. This is the preferred solution.
- Modify the **Device** resource of your **bacula-sd.conf** file to include:

  ```
  Hardware End of File = no
  ```

  This will cause Bacula to use the MTFSF request to seek to the end of the medium, and Bacula will keep track of the file number itself.

37.3.4 Incorrect Number of Blocks or Positioning Errors

Bacula's preferred method of working with tape drives (sequential devices) is to run in variable block mode, and this is what is set by default. You should first ensure that your tape drive is set for variable block mode (see below).

If your tape drive is in fixed block mode and you have told Bacula to use different fixed block sizes or variable block sizes (default), you will get errors when Bacula attempts to forward space to the correct block (the kernel driver's idea of tape blocks will not correspond to Bacula's).

All modern tape drives support variable tape blocks, but some older drives (in particular the QIC drives) as well as the ATAPI ide-scsi driver run only in fixed block mode. The Travan tape drives also apparently must run in fixed block mode (to be confirmed).

Even in variable block mode, with the exception of the first record on the second or subsequent volume of a multi-volume backup, Bacula will write blocks of a fixed size. However, in reading a tape, Bacula will assume that for each read request, exactly one block from the tape will be transferred. This the most common way that tape drives work and is well supported by **Bacula**.

Drives that run in fixed block mode can cause serious problems for Bacula if the drive's block size does not correspond exactly to **Bacula's** block size. In fixed block size mode, drivers may transmit a partial block or multiple blocks for a single read request. From **Bacula's** point of view, this destroys the concept of tape blocks. It is much better to run in variable block mode, and almost all modern drives (the OnStream is an exception) run in variable block mode. In order for Bacula to run in fixed block mode, you must include the following records in the Storage daemon's Device resource definition:

```
Minimum Block Size = nnn
Maximum Block Size = nnn
```

where **nnn** must be the same for both records and must be identical to the driver's fixed block size.

We recommend that you avoid this configuration if at all possible by using variable block sizes.

If you must run with fixed size blocks, make sure they are not 512 bytes. This is too small and the overhead that Bacula has with each record will become excessive. If at all possible set any fixed block size to something like 64,512 bytes or possibly 32,768 if 64,512 is too large for your drive. See below for the details on checking and setting the default drive block size.

To recover files from tapes written in fixed block mode, see below.

37.3.5 Ensuring that the Tape Modes Are Properly Set – Linux Only

If you have a modern SCSI tape drive and you are having problems with the **test** command as noted above, it may be that some program has set one or more of your SCSI driver's options to non-default values. For example, if your driver is set to work in SysV manner, Bacula will not work correctly because it expects BSD behavior. To reset your tape drive to the default values, you can try the following, but **ONLY** if you have a SCSI tape drive on a **Linux** system:

```
become super user
```

```
mt -f /dev/nst0 rewind
mt -f /dev/nst0 stoptions buffer-writes async-writes read-ahead
```

The above commands will clear all options and then set those specified. None of the specified options are required by Bacula, but a number of other options such as SysV behavior must not be set. Bacula does not support SysV tape behavior. On systems other than Linux, you will need to consult your **mt** man pages or documentation to figure out how to do the same thing. This should not really be necessary though – for example, on both Linux and Solaris systems, the default tape driver options are compatible with Bacula. On Solaris systems, you must take care to specify the correct device name on the **Archive device** directive. See above for more details.

You may also want to ensure that no prior program has set the default block size, as happened to one user, by explicitly turning it off with:

```
mt -f /dev/nst0 defblksize 0
```

If you are running a Linux system, and the above command does not work, it is most likely because you have not loaded the appropriate **mt** package, which is often called **mt_st**, but may differ according to your distribution.

If you would like to know what options you have set before making any of the changes noted above, you can now view them on Linux systems, thanks to a tip provided by Willem Riede. Do the following:

```
become super user
mt -f /dev/nst0 stsetoptions 0
grep st0 /var/log/messages
```

and you will get output that looks something like the following:

```
kernel: st0: Mode 0 options: buffer writes: 1, async writes: 1, read ahead: 1
kernel: st0:    can bsr: 0, two FMs: 0, fast mteom: 0, auto lock: 0,
kernel: st0:    defs for wr: 0, no block limits: 0, partitions: 0, s2 log: 0
kernel: st0:    sysv: 0 nowait: 0
```

Note, I have chopped off the beginning of the line with the date and machine name for presentation purposes.

Some people find that the above settings only last until the next reboot, so please check this otherwise you may have unexpected problems.

Beginning with Bacula version 1.35.8, if Bacula detects that you are running in variable block mode, it will attempt to set your drive appropriately. All OSes permit setting variable block mode, but some OSes do not permit setting the other modes that Bacula needs to function properly.

37.3.6 Tape Hardware Compression and Blocking Size

As far as I can tell, there is no way with the **mt** program to check if your tape hardware compression is turned on or off. You can, however, turn it on by using (on Linux):

```
become super user
mt -f /dev/nst0 defcompression 1
```

and of course, if you use a zero instead of the one at the end, you will turn it off.

If you have built the **mtx** program in the **depkgs** package, you can use tapeinfo to get quite a bit of information about your tape drive even if it is not an autochanger. This program is called using the SCSI control device. On Linux for tape drive /dev/nst0, this is usually /dev/sg0, while on FreeBSD for /dev/nsa0, the control device is often /dev/pass2. For example on my DDS-4 drive (/dev/nst0), I get the following:

```
tapeinfo -f /dev/sg0
Product Type: Tape Drive
Vendor ID: 'HP      '
Product ID: 'C5713A          '
Revision: 'H107'
Attached Changer: No
MinBlock:1
MaxBlock:16777215
SCSI ID: 5
SCSI LUN: 0
Ready: yes
BufferedMode: yes
Medium Type: Not Loaded
Density Code: 0x26
BlockSize: 0
```

where the **DataCompEnabled: yes** means that tape hardware compression is turned on. You can turn it on and off (yes—no) by using the **mt** commands given above. Also, this output will tell you if the **BlockSize** is non-zero and hence set for a particular block size. Bacula is not likely to work in such a situation because it will normally attempt to write blocks of 64,512 bytes, except the last block of the job which will generally be shorter. The first thing to try is setting the default block size to zero using the **mt -f /dev/nst0 defblksize 0** command as shown above. On FreeBSD, this would be something like: **mt -f /dev/nsa0 blocksize 0**.

On some operating systems with some tape drives, the amount of data that can be written to the tape and whether or not compression is enabled is determined by the density usually the **mt -f /dev/nst0 setdensity xxx** command. Often **mt -f /dev/nst0 status** will print out the current density code that is used with the drive. Most systems, but unfortunately not all, set the density to the maximum by default. On some systems, you can also get a list of all available density codes with: **mt -f /dev/nst0 densities** or a similar **mt** command. Note, for DLT and SDLT devices, no-compression versus compression is very often controlled by the density code. On FreeBSD systems, the compression mode is set using **mt -f /dev/nsa0 comp xxx** where xxx is the mode you want. In general, see **man mt** for the options available on your system.

Note, some of the above **mt** commands may not be persistent depending on your system configuration. That is they may be reset if a program other than Bacula uses the drive or, as is frequently the case, on reboot of your system.

If your tape drive requires fixed block sizes (very unusual), you can use the following records:

```
Minimum Block Size = nnn
Maximum Block Size = nnn
```

in your Storage daemon's Device resource to force Bacula to write fixed size blocks (where you sent nnn to be the same for both of the above records). This should be done only if your drive does not support variable block sizes, or you have some other strong reasons for using fixed block sizes. As mentioned above, a small fixed block size of 512 or 1024 bytes will be very inefficient. Try to set any fixed block size to something like 64,512 bytes or larger if your drive will support it.

Also, note that the **Medium Type** field of the output of **tapeinfo** reports **Not Loaded**, which is not correct. As a consequence, you should ignore that field as well as the **Attached Changer** field.

To recover files from tapes written in fixed block mode, see below.

37.3.7 Tape Modes on FreeBSD

On most FreeBSD systems such as 4.9 and most tape drives, Bacula should run with:

```
mt  -f  /dev/nsa0  seteotmodel  2
mt  -f  /dev/nsa0  blocksize   0
mt  -f  /dev/nsa0  comp  enable
```

You might want to put those commands in a startup script to make sure your tape driver is properly initialized before running Bacula, because depending on your system configuration, these modes may be reset if a program other than Bacula uses the drive or when your system is rebooted.

Then according to what the **btape test** command returns, you will probably need to set the following (see below for an alternative):

```
  Hardware End of Medium = no
  BSF at EOM = yes
  Backward Space Record = no
  Backward Space File = no
  Fast Forward Space File = no
  TWO EOF = yes
```

Then be sure to run some append tests with Bacula where you start and stop Bacula between appending to the tape, or use **btape** version 1.35.1 or greater, which includes simulation of stopping/restarting Bacula.

Please see the file **platforms/freebsd/pthreads-fix.txt** in the main Bacula directory concerning **important** information concerning compatibility of Bacula and your system. A much more optimal Device configuration is shown below, but does not work with all tape drives. Please test carefully before putting either into production.

Note, for FreeBSD 4.10-RELEASE, using a Sony TSL11000 L100 DDS4 with an autochanger set to variable block size and DCLZ compression, Brian McDonald reports that to get Bacula to append correctly between Bacula executions, the correct values to use are:

```
mt  -f  /dev/nsa0  seteotmodel  1
mt  -f  /dev/nsa0  blocksize  0
mt  -f /dev/nsa0  comp  enable
```

and

```
  Hardware End of Medium = no
  BSF at EOM = no
  Backward Space Record = no
  Backward Space File = no
  Fast Forward Space File = yes
  TWO EOF = no
```

This has been confirmed by several other people using different hardware. This configuration is the preferred one because it uses one EOF and no backspacing at the end of the tape, which works much more efficiently and reliably with modern tape drives.

Finally, here is a Device configuration that Danny Butroyd reports to work correctly with the Overland Powerloader tape library using LT0-2 and FreeBSD 5.4-Stable:

```
# Overland Powerloader LT02 - 17 slots single drive
Device {
  Name = Powerloader
  Media Type = LTO-2
  Archive Device = /dev/nsa0
  AutomaticMount = yes;
  AlwaysOpen = yes;
  RemovableMedia = yes;
  RandomAccess = no;
  Changer Command = "/usr/local/sbin/mtx-changer %c %o %S %a %d"
  Changer Device = /dev/pass2
  AutoChanger = yes
  Alert Command = "sh -c 'tapeinfo -f %c |grep TapeAlert|cat'"

  # FreeBSD Specific Settings
  Offline On Unmount = no
  Hardware End of Medium = no
  BSF at EOM = yes
  Backward Space Record = no
  Fast Forward Space File = no
  TWO EOF = yes
}

The following Device resource works fine with Dell PowerVault 110T and
120T devices on both FreeBSD 5.3 and on NetBSD 3.0.  It also works
with Sony AIT-2 drives on FreeBSD.
\footnotesize
\begin{verbatim}
Device {
  ...
  # FreeBSD/NetBSD Specific Settings
  Hardware End of Medium = no
  BSF at EOM = yes
  Backward Space Record = no
  Fast Forward Space File = yes
```

```
  TWO EOF = yes
}
```

On FreeBSD version 6.0, it is reported that you can even set Backward Space Record = yes.

37.3.8 Finding your Tape Drives and Autochangers on FreeBSD

On FreeBSD, you can do a **camcontrol devlist** as root to determine what drives and autochangers you have. For example,

```
undef# camcontrol devlist
    at scbus0 target 2 lun 0 (pass0,sa0)
    at scbus0 target 4 lun 0 (pass1,sa1)
    at scbus0 target 4 lun 1 (pass2)
```

from the above, you can determine that there is a tape drive on **/dev/sa0** and another on **/dev/sa1** in addition since there is a second line for the drive on **/dev/sa1**, you know can assume that it is the control device for the autochanger (i.e. **/dev/pass2**). It is also the control device name to use when invoking the tapeinfo program. E.g.

```
tapeinfo -f /dev/pass2
```

37.3.9 Using the OnStream driver on Linux Systems

Bacula version 1.33 (not 1.32x) is now working and ready for testing with the OnStream kernel osst driver version 0.9.14 or above. Osst is available from: http://sourceforge.net/projects/osst/.

To make Bacula work you must first load the new driver then, as root, do:

```
  mt -f /dev/nosst0 defblksize 32768
```

Also you must add the following to your Device resource in your Storage daemon's conf file:

```
 Minimum Block Size = 32768
 Maximum Block Size = 32768
```

Here is a Device specification provided by Michel Meyers that is known to work:

```
Device {
  Name = "Onstream DI-30"
  Media Type = "ADR-30"
  Archive Device = /dev/nosst0
  Minimum Block Size = 32768
  Maximum Block Size = 32768
  Hardware End of Medium = yes
  BSF at EOM = no
  Backward Space File = yes
  Fast Forward Space File = yes
  Two EOF = no
  AutomaticMount = yes
  AlwaysOpen = yes
  Removable Media = yes
}
```

37.4 Hardware Compression on EXB-8900

To active, check, or disable the hardware compression feature on an EXB-8900, use the exabyte MammothTool. You can get it here: http://www.exabyte.com/support/online/downloads/index.cfm. There is a Solaris version of this tool. With option -C 0 or 1 you can disable or activate compression. Start this tool without any options for a small reference.

37.4.1 Using btape to Simulate Filling a Tape

Because there are often problems with certain tape drives or systems when end of tape conditions occur, **btape** has a special command **fill** that causes it to write random data to a tape until the tape fills. It then writes at least one more Bacula block to a second tape. Finally, it reads back both tapes to ensure that the data has been written in a way that Bacula can recover it. Note, there is also a single tape option as noted below, which you should use rather than the two tape test. See below for more details.

This can be an extremely time consuming process (here it is about 6 hours) to fill a full tape. Note, that btape writes random data to the tape when it is filling it. This has two consequences: 1. it takes a bit longer to generate the data, especially on slow CPUs. 2. the total amount of data is approximately the real physical capacity of your tape, regardless of whether or not the tape drive compression is on or off. This is because random data does not compress very much.

To begin this test, you enter the **fill** command and follow the instructions. There are two options: the simple single tape option and the multiple tape option. Please use only the simple single tape option because the multiple tape option still doesn't work totally correctly. If the single tape option does not succeed, you should correct the problem before using Bacula.

37.5 Recovering Files Written With Fixed Block Sizes

If you have been previously running your tape drive in fixed block mode (default 512) and Bacula with variable blocks (default), then in version 1.32f-x and 1.34 and above, Bacula will fail to recover files because it does block spacing, and because the block sizes don't agree between your tape drive and Bacula it will not work.

The long term solution is to run your drive in variable block mode as described above. However, if you have written tapes using fixed block sizes, this can be a bit of a pain. The solution to the problem is: while you are doing a restore command using a tape written in fixed block size, ensure that your drive is set to the fixed block size used while the tape was written. Then when doing the **restore** command in the Console program, do not answer the prompt **yes/mod/no**. Instead, edit the bootstrap file (the location is listed in the prompt) using any ASCII editor. Remove all **VolBlock** lines in the file. When the file is re-written, answer the question, and Bacula will run without using block positioning, and it should recover your files.

37.6 Tape Blocking Modes

SCSI tapes may either be written in **variable** or **fixed** block sizes. Newer drives support both modes, but some drives such as the QIC devices always use fixed block sizes. Bacula attempts to fill and write complete blocks (default 65K), so that in normal mode (variable block size), Bacula will always write blocks of the same size except the last block of a Job. If Bacula is configured to write fixed block sizes, it will pad the last block of the Job to the correct size. Bacula expects variable tape block size drives to behave as follows: Each write to the drive results in a single record being written to the tape. Each read returns a single record. If you request less bytes than are in the record, only those number of bytes will be returned, but the entire logical record will have been read (the next read will retrieve the next record). Thus data from a single write is always returned in a single read, and sequentially written records are returned by sequential reads.

Bacula expects fixed block size tape drives to behave as follows: If a write length is greater than the physical block size of the drive, the write will be written as two blocks each of the fixed physical size. This single write may become multiple physical records on the tape. (This is not a good situation). According to the documentation, one may never write an amount of data that is not the exact multiple of the blocksize (it is not specified if an error occurs or if the the last record is padded). When reading, it is my understanding that each read request reads one physical record from the tape. Due to the complications of fixed block size tape drives, you should avoid them if possible with Bacula, or you must be ABSOLUTELY certain that you use fixed block sizes within Bacula that correspond to the physical block size of the tape drive. This will ensure that Bacula has a one to one correspondence between what it writes and the physical record on the tape.

Please note that Bacula will not function correctly if it writes a block and that block is split into two or more physical records on the tape. Bacula assumes that each write causes a single record to be written, and that it can sequentially recover each of the blocks it has written by using the same number of sequential reads as it had written.

37.7 Details of Tape Modes

Rudolf Cejka has provided the following information concerning certain tape modes and MTEOM.

Tape level It is always possible to position filemarks or blocks, whereas positioning to the end-of-data is only optional feature, however it is implemented very often. SCSI specification also talks about optional sequential filemarks, setmarks and sequential setmarks, but these are not implemented so often. Modern tape drives keep track of file positions in built-in chip (AIT, LTO) or at the beginning of the tape (SDLT), so there is not any speed difference, if end-of-data or filemarks is used (I have heard, that LTO-1 from all 3 manufacturers do not use its chip for file locations, but a tape as in SDLT case, and I'm not sure about LTO-2 and LTO-3 case). However there is a big difference, that end-of-data ignores file position, whereas filemarks returns the real number of skipped files, so OS can track current file number just in filemarks case.

OS level Solaris does use just SCSI SPACE Filemarks, it does not support SCSI SPACE End-of-data. When MTEOM is called, Solaris does use

SCSI SPACE Filemarks with count = 1048576 for fast mode, and combination of SCSI SPACE Filemarks with count = 1 with SCSI SPACE Blocks with count = 1 for slow mode, so EOD mark on the tape on some older tape drives is not skipped. File number is always tracked for MTEOM.

Linux does support both SCSI SPACE Filemarks and End-of-data: When MTEOM is called in MT_ST_FAST_MTEOM mode, SCSI SPACE End-of-data is used. In the other case, SCSI SPACE Filemarks with count = 8388607 is used. There is no real slow mode like in Solaris - I just expect, that for older tape drives Filemarks may be slower than End-of-data, but not so much as in Solaris slow mode. File number is tracked for MTEOM just without MT_ST_FAST_MTEOM - when MT_ST_FAST_MTEOM is used, it is not.

FreeBSD does support both SCSI SPACE Filemarks and End-of-data, but when MTEOD (MTEOM) is called, SCSI SPACE End-of-data is always used. FreeBSD never use SCSI SPACE Filemarks for MTEOD. File number is never tracked for MTEOD.

Bacula level When **Hardware End of Medium = Yes** is used, MTEOM is called, but it does not mean, that hardware End-of-data must be used. When Hardware End of Medium = No, if Fast Forward Space File = Yes, MTFSF with count = 32767 is used, else Block Read with count = 1 with Forward Space File with count = 1 is used, which is really very slow.

Hardware End of Medium = Yes—No The name of this option is misleading and is the source of confusion, because it is not the hardware EOM, what is really switched here.

If I use Yes, OS must not use SCSI SPACE End-of-data, because Bacula expects, that there is tracked file number, which is not supported by SCSI specification. Instead, the OS have to use SCSI SPACE Filemarks.

If I use No, an action depends on Fast Forward Space File.

When I set **Hardware End of Medium = no** and **Fast Forward Space File = no** file positioning was very slow on my LTO-3 (about ten to 100 minutes), but

with **Hardware End of Medium = no** and **Fast Forward Space File = yes**, the time is ten to 100 times faster (about one to two minutes).

37.8 Autochanger Errors

If you are getting errors such as:

```
3992 Bad autochanger "load slot 1, drive 1": ERR=Child exited with code 1.
```

and you are running your Storage daemon as non-root, then most likely you are having permissions problems with the control channel. Running as root, set permissions on /dev/sgX so that the userid and group of your Storage daemon can access the device. You need to ensure that you all access to the proper control device, and if you don't have any SCSI disk drives (including SATA drives), you might want to change the permissions on /dev/sg*.

37.9 Syslog Errors

If you are getting errors such as:

```
: kernel: st0: MTSETDRVBUFFER only allowed for root
```

you are most likely running your Storage daemon as non-root, and Bacula is attempting to set the correct OS buffering to correspond to your Device resource. Most OSes allow only root to issue this ioctl command. In general, the message can be ignored providing you are sure that your OS parameters are properly configured as described earlier in this manual. If you are running your Storage daemon as root, you should not be getting these system log messages, and if you are, something is probably wrong.

Chapter 38

What To Do When Bacula Crashes (Kaboom)

If you are running on a Linux system, and you have a set of working configuration files, it is very unlikely that **Bacula** will crash. As with all software, however, it is inevitable that someday, it may crash, particularly if you are running on another operating system or using a new or unusual feature.

This chapter explains what you should do if one of the three **Bacula** daemons (Director, File, Storage) crashes. When we speak of crashing, we mean that the daemon terminates abnormally because of an error. There are many cases where Bacula detects errors (such as PIPE errors) and will fail a job. These are not considered crashes. In addition, under certain conditions, Bacula will detect a fatal in the configuration, such as lack of permission to read/write the working directory. In that case, Bacula will force itself to crash with a SEGFAULT. However, before crashing, Bacula will normally display a message indicating why. For more details, please read on.

38.1 Traceback

Each of the three Bacula daemons has a built-in exception handler which, in case of an error, will attempt to produce a traceback. If successful the traceback will be emailed to you.

For this to work, you need to ensure that a few things are setup correctly on your system:

1. You must have a version of Bacula built with debug information turned on and not stripped of debugging symbols.
2. You must have an installed copy of **gdb** (the GNU debugger), and it must be on **Bacula's** path. On some systems such as Solaris, **gdb** may be replaced by **dbx**.
3. The Bacula installed script file **btraceback** must be in the same directory as the daemon which dies, and it must be marked as executable.
4. The script file **btraceback.gdb** must have the correct path to it specified in the **btraceback** file.
5. You must have a **mail** program which is on **Bacula's** path. By default, this **mail** program is set to **bsmtp**, so it must be correctly configured.
6. If you run either the Director or Storage daemon under a non-root userid, you will most likely need to modify the **btraceback** file to do something like **sudo** (raise to root priority) for the call to **gdb** so that it has the proper permissions to debug Bacula.

If all the above conditions are met, the daemon that crashes will produce a traceback report and email it to you. If the above conditions are not true, you can either run the debugger by hand as described below, or you may be able to correct the problems by editing the **btraceback** file. I recommend not spending too much time on trying to get the traceback to work as it can be very difficult.

The changes that might be needed are to add a correct path to the **gdb** program, correct the path to the **btraceback.gdb** file, change the **mail** program or its path, or change your email address. The key line in the **btraceback** file is:

```
gdb -quiet -batch -x /home/kern/bacula/bin/btraceback.gdb \
     $1 $2 2>\&1 | bsmtp -s "Bacula traceback" your-address@xxx.com
```

Since each daemon has the same traceback code, a single btraceback file is sufficient if you are running more than one daemon on a machine.

38.2 Testing The Traceback

To "manually" test the traceback feature, you simply start **Bacula** then obtain the **PID** of the main daemon thread (there are multiple threads).

The output produced here will look different depending on what OS and what version of the kernel you are running. Unfortunately, the output had to be split to fit on this page:

```
[kern@rufus kern]$ ps fax --columns 132 | grep bacula-dir
 2103 ?        S      0:00 /home/kern/bacula/k/src/dird/bacula-dir -c
                                       /home/kern/bacula/k/src/dird/dird.conf
 2104 ?        S      0:00  \_ /home/kern/bacula/k/src/dird/bacula-dir -c
                                       /home/kern/bacula/k/src/dird/dird.conf
 2106 ?        S      0:00      \_ /home/kern/bacula/k/src/dird/bacula-dir -c
                                       /home/kern/bacula/k/src/dird/dird.conf
 2105 ?        S      0:00      \_ /home/kern/bacula/k/src/dird/bacula-dir -c
                                       /home/kern/bacula/k/src/dird/dird.conf
```

which in this case is 2103. Then while Bacula is running, you call the program giving it the path to the Bacula executable and the **PID**. In this case, it is:

```
./btraceback /home/kern/bacula/k/src/dird 2103
```

It should produce an email showing you the current state of the daemon (in this case the Director), and then exit leaving **Bacula** running as if nothing happened. If this is not the case, you will need to correct the problem by modifying the **btraceback** script.

Typical problems might be that **gdb** or **dbx** for Solaris is not on the default path. Fix this by specifying the full path to it in the **btraceback** file. Another common problem is that you haven't modified the script so that the **bsmtp** program has an appropriate smtp server or the proper syntax for your smtp server. If you use the **mail** program and it is not on the default path, it will also fail. On some systems, it is preferable to use **Mail** rather than **mail**.

38.3 Getting A Traceback On Other Systems

It should be possible to produce a similar traceback on systems other than Linux, either using **gdb** or some other debugger. Solaris with **dbx** loaded works quite fine. On other systems, you will need to modify the **btraceback** program to invoke the correct debugger, and possibly correct the **btraceback.gdb** script to have appropriate commands for your debugger. If anyone succeeds in making this work with another debugger, please send us a copy of what you modified. Please keep in mind that for any debugger to work, it will most likely need to run as root, so you may need to modify the **btraceback** script accordingly.

38.4 Manually Running Bacula Under The Debugger

If for some reason you cannot get the automatic traceback, or if you want to interactively examine the variable contents after a crash, you can run Bacula under the debugger. Assuming you want to run the Storage daemon under the debugger (the technique is the same for the other daemons, only the name changes), you would do the following:

1. Start the Director and the File daemon. If the Storage daemon also starts, you will need to find its PID as shown above (ps fax — grep bacula-sd) and kill it with a command like the following:

   ```
   kill -15 PID
   ```

   where you replace **PID** by the actual value.

2. At this point, the Director and the File daemon should be running but the Storage daemon should not.

3. cd to the directory containing the Storage daemon

4. Start the Storage daemon under the debugger:

   ```
   gdb ./bacula-sd
   ```

5. Run the Storage daemon:

   ```
   run -s -f -c ./bacula-sd.conf
   ```

   You may replace the **./bacula-sd.conf** with the full path to the Storage daemon's configuration file.

6. At this point, Bacula will be fully operational.

7. In another shell command window, start the Console program and do what is necessary to cause Bacula to die.

8. When Bacula crashes, the **gdb** shell window will become active and **gdb** will show you the error that occurred.

9. To get a general traceback of all threads, issue the following command:

```
thread apply all bt
```

   After that you can issue any debugging command.

38.5 Getting Debug Output from Bacula

Each of the daemons normally has debug compiled into the program, but disabled. There are two ways to enable the debug output. One is to add the **-d nnn** option on the command line when starting the debugger. The **nnn** is the debug level, and generally anything between 50 and 200 is reasonable. The higher the number, the more output is produced. The output is written to standard output.

The second way of getting debug output is to dynamically turn it on using the Console using the **setdebug** command. The full syntax of the command is:

```
setdebug level=nnn client=client-name storage=storage-name dir
```

If none of the options are given, the command will prompt you. You can selectively turn on/off debugging in any or all the daemons (i.e. it is not necessary to specify all the components of the above command).

Chapter 39

The Windows Version of Bacula

At the current time only the File daemon or Client program has been thouroughly tested on Windows and is suitable for 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.

As of Bacula version 1.39.20 or greater, the installer is capable of installing not just the Client program, but also the Director and the Storage daemon and all the other programs that were previously available only on Unix systems. These additional programs, notably the Director and Storage daemon, have been partially tested, are reported to have some bugs, and still need to be documented. They are not yet supported, and we cannot currently accept or fix bug reports on them. Consequently, please test them carefully before putting them into a critical production environment.

The Windows version of the Bacula File daemon has been tested on Win98, WinMe, WinNT, WinXP, Win2000, and Windows 2003 systems. We have coded to support Win95, but no longer have a system for testing. The Windows version of Bacula is a native Win32 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 and displays its icon in the system icon tray, and provides a system tray menu to obtain additional information on how Bacula is running (status and events dialog boxes). If so desired, it can also be stopped by using the system tray menu, though this should normally never be necessary.

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.

39.1 Win32 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.

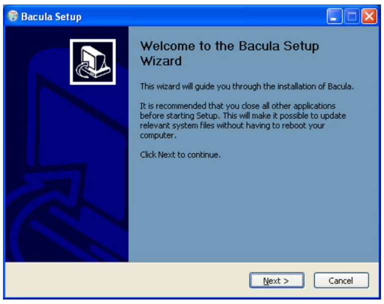
If you have a previous version Bacula (1.39.20 or lower) installed, you should stop the service, uninstall it, and remove the Bacula installation directory possibly saving your bacula-fd.conf, bconsole.conf, and bwx-console.conf files for use with the new version you will install. The Uninstall program is normally found in **c:\bacula\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 new installer (1.39.22 and later) 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.

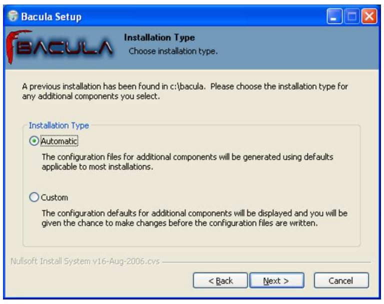
Finally, proceed with the installation.

- 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 **winbacula-1.xx.0.exe** NSIS install icon. The actual name of the icon will vary from one release version to another.

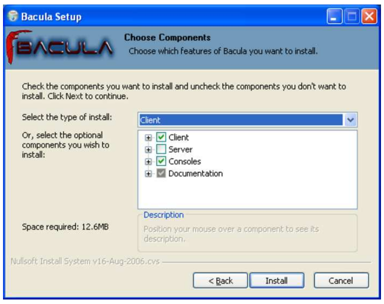
  
  winbacula-1.xx.0.exe
- Once launched, the installer wizard will ask you if you want to install Bacula.



- Next you will be asked to select the installation type.

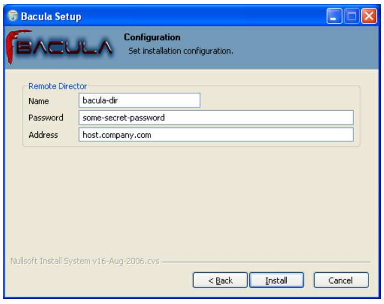


- If you proceed, you will be asked to select the components to be installed. You may install the Bacula program (Bacula File Service) and or the documentation. Both will be installed in sub-directories of the install location that you choose later. The components dialog looks like the following:

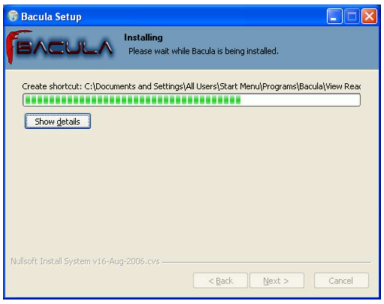


- If you are installing for the first time, you will be asked to enter some very basic information about your configuration. If you are not sure what to enter, or have previously saved configuration files, you can put anything you want into the fields, then either replace the configuration files later with the ones saved, or edit the file.

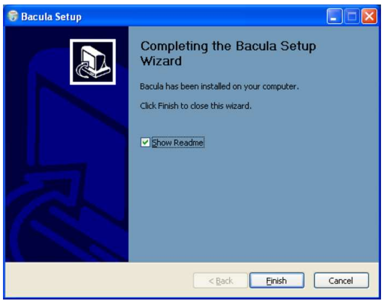
  If you are upgrading an existing installation, the following will not be displayed.

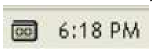


- While the various files are being loaded, you will see the following dialog:



- Finally, the finish dialog will appear:



That should complete the installation process. 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 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.

39.2 Post Win32 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 (ctl-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.

39.3 Uninstalling Bacula on Win32

Once Bacula has been installed, it can be uninstalled using the standard Windows Add/Remove Programs dialog found on the Control panel.

39.4 Dealing with Win32 Problems

Sometimes Win32 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: http://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: http://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 (Win32, 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\bin\bacula-fd -t >out
type out
```

The precise path to bacula-fd depends on where it is installed. The example above is the default used in 1.39.22 and later. 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. In later versions of Bacula (1.34 on, I think), 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.

39.5 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. For version 1.37.25 and greater, 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.

In Bacula version 1.31 and later, we use 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 on WinNT/2K/XP 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 WinNT/2K/XP 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 WinNT using the native API calls (BackupRead/BackupWrite) cannot be restored on Win95/98/Me or Unix systems. In principle, a file backed up on WinNT can be restored on WinXP, but this remains to be seen in practice (not yet tested). In addition, the stand-alone tools such as **bls** and **bextract** cannot be used to retrieve the data for those files because those tools are not available on Windows. All restores must use the Bacula **restore** command. As of Bacula 1.39.x, thanks to Thorsten Engel, this restriction is removed, and 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 WinNT/2K/XP system and restore it on a Unix/Win95/98/Me 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 systems, such as WinNT/2K/XP, you may have to reset the ownership of such restored files. Any file backed up on WinNT/2K/XP should in principle be able to be restored to a similar system (i.e. WinNT/2K/XP), however, I am unsure of the consequences if the owner information and accounts are not identical on both systems. Bacula will not let you restore files backed up on WinNT/2K/XP to any other system (i.e. Unix Win95/98/Me) if you have used the defaults.

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 WinNT/2K/XP 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:

Backup OS	Restore OS	Results
WinMe	WinMe	Works
WinMe	WinNT	Works (SYSTEM permissions)
WinMe	WinXP	Works (SYSTEM permissions)
WinMe	Linux	Works (SYSTEM permissions)
WinXP	WinXP	Works
WinXP	WinNT	Works (all files OK, but got "The data is invalid" message)
WinXP	WinMe	Error: Win32 data stream not supported.
WinXP	WinMe	Works if **Portable=yes** specified during backup.
WinXP	Linux	Error: Win32 data stream not supported.
WinXP	Linux	Works if **Portable=yes** specified during backup.
WinNT	WinNT	Works
WinNT	WinXP	Works
WinNT	WinMe	Error: Win32 data stream not supported.
WinNT	WinMe	Works if **Portable=yes** specified during backup.
WinNT	Linux	Error: Win32 data stream not supported.

WinNT	Linux	Works if **Portable=yes** specified during backup.
Linux	Linux	Works
Linux	WinNT	Works (SYSTEM permissions)
Linux	WinMe	Works
Linux	WinXP	Works (SYSTEM permissions)

Note: with Bacula versions 1.39.x and later, non-portable Windows data can be restore to any machine.

39.6 Volume Shadow Copy Service

In version 1.37.30 and greater, you can turn on Microsoft's Volume Shadow Copy Service (VSS).

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.

Bacula supports VSS on both Windows 2003 and Windows XP. 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 - 3
    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/n
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.

39.7 VSS Problems

Problems!VSS

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).

39.8 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.

39.9 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 Win32 program, so you must find and download it from the Internet.

39.10 Windows Disaster Recovery

We don't currently have a good solution for disaster recovery on Windows as we do on Linux. The main piece lacking 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.

Please see Disaster Recovery of Win32 Systems for the latest suggestion, which looks very promising.

It looks like Bart PE Builder, which creates a Windows PE (Pre-installation Environment) Boot-CD, may be just what is needed to build a complete disaster recovery system for Win32. This distribution can be found at http://www.nu2.nu/pebuilder/.

39.11 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.

39.12 Windows Ownership and Permissions Problems

If you restore files backed up from WinNT/XP/2K 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 Win32 permissions is the program **SetACL**, which can be found at http://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.

39.13 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 Win32 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:

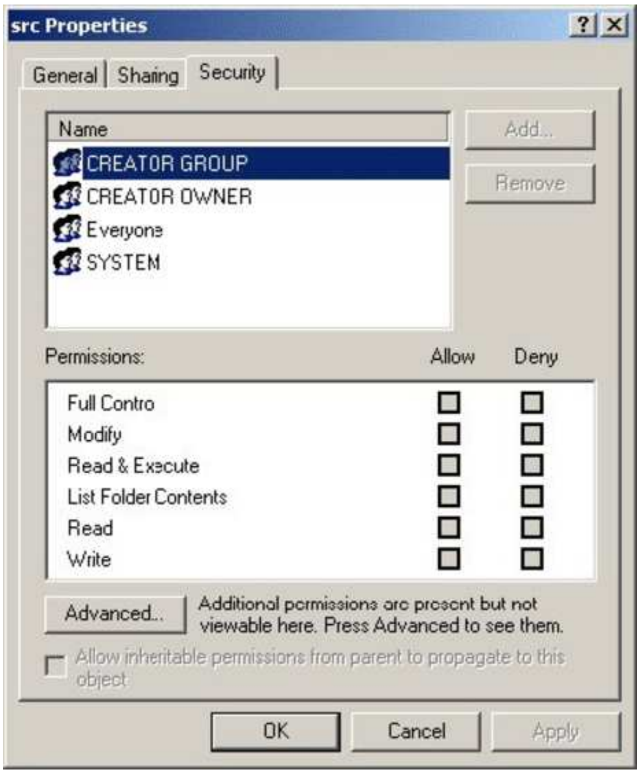
If you encounter this message, the following steps will change the permissions to allow full access.

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**.

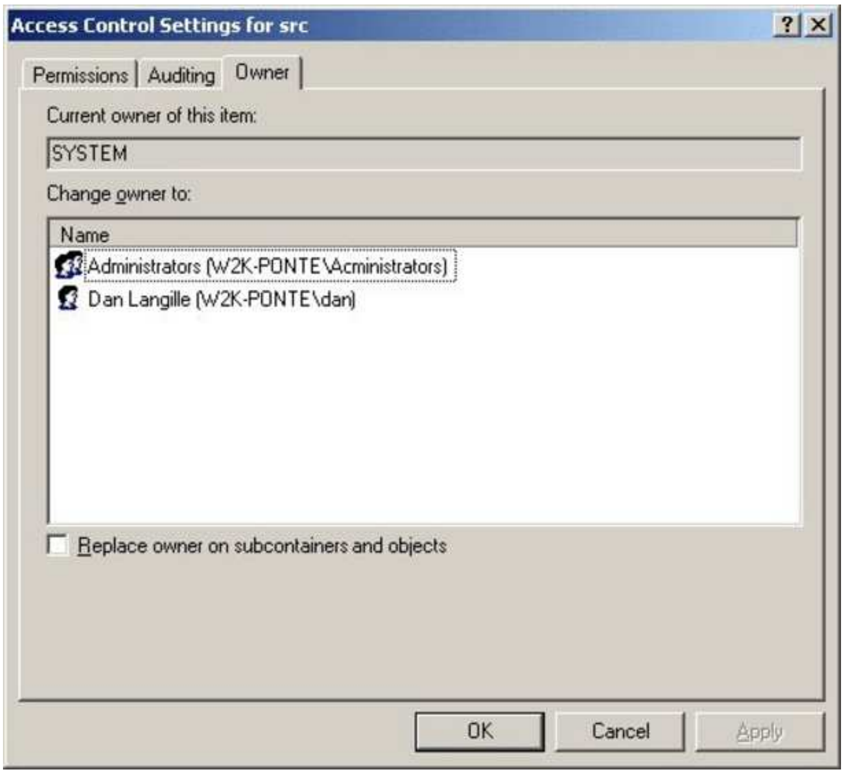
You should see something like this:



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

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.

39.14 Backing Up the WinNT/XP/2K System State

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:\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.

39.15 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.

Bacula versions prior to 1.37.28 do not support Windows Unicode filenames. As of that version, both **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 beginning with Bacula version 1.39.20. Older Bacula versions support only 260 character path/filenames.

39.16 Win32 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.

39.17 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 or one from the Sysinternals project.

Chapter 40

Disaster Recovery Using Bacula

40.1 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. As a consequence, this chapter will discuss in detail disaster recovery (also called Bare Metal Recovery) for **Linux** and **Solaris**. For Solaris, the procedures are still quite manual. For FreeBSD the same procedures may be used but they are not yet developed. For Win32, a number of Bacula users have reported success using BartPE.

40.2 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?

40.3 Steps to Take Before Disaster Strikes

- Create a rescue or CDROM for each of your Linux systems. Generally, they are offered by each distribution, and there are many good rescue disks on the Web (Knoppix, sysrescuecd, PLD Linux rescue CD, tomsrtbt, RIP ...

- Create a bacula-hostname directory on each machine and save it somewhere – possibly on a USB key.

- Ensure that you always have a valid bootstrap file for your backup and that it is saved to an alternate machine. This will permit you to easily do a full restore of your system.

- If possible copy your catalog nightly to an alternate machine. If you have a valid bootstrap file, this is not necessary, but can be very useful if you do not want to reload everything. .

- Ensure that you always have a valid bootstrap file for your catalog backup that is saved to an alternate machine. This will permit you to restore your catalog more easily if needed.

- Test using the Rescue CDROM before you are forced to use it in an emergency situation.

- Make a copy of your Bacula .conf files, particularly your bacula-dir.conf, and your bacula-sd.conf files, because if your server goes down, these files will be needed to get it back up and running, and they can be difficult to rebuild from memory.

40.4 Bare Metal Recovery on Linux with a Rescue CD

As an alternative to creating a Rescue CD, please see the section below entitled Bare Metal Recovery using a LiveCD.

Bacula previously had a Rescue CD. Unfortunately, this CD did not work on every Linux Distro, and in addition, Linux is evolving with different boot methods, more and more complex hardware configurations (LVM, RAID, WiFi, USB, ...). As a consequence, the Bacula Rescue CD as it was originally envisioned no longer exists.

However there are many other good rescue disks available. A so called "Bare Metal" recovery is one where you start with an empty hard disk and you restore your machine. There are also cases where you may lose a file or a directory and want it restored. Please see the previous chapter for more details for those cases.

Bare Metal Recovery assumes that you have the following items for your system:

- A Rescue CDROM containing a copy of your OS.
- Perhaps a copy of your hard disk information, as well as a statically linked version of the Bacula File daemon.
- A full Bacula backup of your system possibly including Incremental or Differential backups since the last Full backup
- A second system running the Bacula Director, the Catalog, and the Storage daemon. (this is not an absolute requirement, but how to get around it is not yet documented here)

40.5 Requirements

40.6 Restoring a Client System

Now, let's assume that your hard disk has just died and that you have replaced it with an new identical drive. In addition, we assume that you have:

1. A recent Bacula backup (Full plus Incrementals)

2. A Rescue CDROM.
3. Your Bacula Director, Catalog, and Storage daemon running on another machine on your local network.

This is a relatively simple case, and later in this chapter, as time permits, we will discuss how you might recover from a situation where the machine that crashes is your main Bacula server (i.e. has the Director, the Catalog, and the Storage daemon).

You will take the following steps to get your system back up and running:

1. Boot with your Rescue CDROM.
2. Start the Network (local network)
3. Re-partition your hard disk(s) as it was before
4. Re-format your partitions
5. Restore the Bacula File daemon (static version)
6. Perform a Bacula restore of all your files
7. Re-install your boot loader
8. Reboot

Now for the details ...

40.7 Boot with your Rescue CDROM

Each rescue disk boots somewhat differently. Please see the instructions that go with your CDROM.

Start the Network: You can test it by pinging another machine, or pinging your broken machine machine from another machine. Do not proceed until your network is up.

Partition Your Hard Disk(s):

Format Your Hard Disk(s):

Mount the Newly Formatted Disks:

Somehow get the static File daemon loaded on your system Put the static file daemon and its conf file in /tmp.

Restore and Start the File Daemon:

```
chroot /mnt/disk /tmp/bacula-fd -c /tmp/bacula-fd.conf
```

The above command starts the Bacula File daemon with the proper root disk location (i.e. **/mnt/disk/tmp**. If Bacula does not start, correct the problem and start it. You can check if it is running by entering:

```
ps fax
```

You can kill Bacula by entering:

```
kill -TERM <pid>
```

where **pid** is the first number printed in front of the first occurrence of **bacula-fd** in the **ps fax** command.

Now, you should be able to use another computer with Bacula installed to check the status by entering:

```
status client=xxxx
```

into the Console program, where xxxx is the name of the client you are restoring.

One common problem is that your **bacula-dir.conf** may contain machine addresses that are not properly resolved on the stripped down system to be restored because it is not running DNS. This is particularly true for the address in the Storage resource of the Director, which may be very well resolved on the Director's machine, but not on the machine being restored and running the File daemon. In that case, be prepared to edit **bacula-dir.conf** to replace the name of the Storage daemon's domain name with its IP address.

Restore Your Files: On the computer that is running the Director, you now run a **restore** command and select the files to be restored (normally everything), but before starting the restore, there is one final change you must make using the **mod** option. You must change the **Where** directory to be the root by using the **mod** option just before running the job and selecting **Where**. Set it to:

```
/
```

then run the restore.

You might be tempted to avoid using **chroot** and running Bacula directly and then using a **Where** to specify a destination of **/mnt/disk**. This is possible, however, the current version of Bacula always restores files to the new location, and thus any soft links that have been specified with absolute paths will end up with **/mnt/disk** prefixed to them. In general this is not fatal to getting your system running, but be aware that you will have to fix these links if you do not use **chroot**.

Final Step:

```
/sbin/grub-install --root-directory=/mnt/disk /dev/hda
```

Note, in this case, you omit the chroot command, and you must replace /dev/hda with your boot device. If you don't know what your boot device is, run the ./run_grub script once and it will tell you.

Finally, I've even run into a case where grub-install was unable to rewrite the boot block. In my case, it produced the following error message:

```
/dev/hdx does not have any corresponding BIOS drive.
```

The solution is to insure that all your disks are properly mounted on /mnt/disk, then do the following:

```
chroot /mnt/disk
mount /dev/pts
```

Then edit the file **/boot/grub/grub.conf** and uncomment the line that reads:

```
#boot=/dev/hda
```

So that it reads:

```
boot=/dev/hda
```

Note, the /dev/hda may be /dev/sda or possibly some other drive depending on your configuration, but in any case, it is the same as the one that you previously tried with **grub-install**.

Then, enter the following commands:

```
grub --batch --device-map=/boot/grub/device.map \
  --config-file=/boot/grub/grub.conf --no-floppy
root (hd0,0)
setup (hd0)
quit
```

If the **grub** call worked, you will get a prompt of **grub**> before the **root**, **setup**, and **quit** commands, and after entering the **setup** command, it should indicate that it successfully wrote the MBR (master boot record).

Reboot: First unmount all your hard disks, otherwise they will not be cleanly shutdown, then reboot your machine by entering **exit** until you get to the main prompt then enter **Ctrl-d**. Once back to the main CDROM prompt, you will need to turn the power off, then back on to your machine to get it to reboot.

If everything went well, you should now be back up and running. If not, re-insert the emergency boot CDROM, boot, and figure out what is wrong.

40.8 Restoring a Server

Above, we considered how to recover a client machine where a valid Bacula server was running on another machine. However, what happens if your server goes down and you no longer have a running Director, Catalog, or Storage daemon? There are several solutions:

1. Bring up static versions of your Director, Catalog, and Storage daemon on the damaged machine.

2. Move your server to another machine.
3. Use a Hot Spare Server on another Machine.

The first option, is very difficult because it requires you to have created a static version of the Director and the Storage daemon as well as the Catalog. If the Catalog uses MySQL or PostgreSQL, this may or may not be possible. In addition, to loading all these programs on a bare system (quite possible), you will need to make sure you have a valid driver for your tape drive.

The second suggestion is probably a much simpler solution, and one I have done myself. To do so, you might want to consider the following steps:

- If you are using MySQL or PostgreSQL, configure, build and install it from source (or use rpms) on your new system.
- Load the Bacula source code onto your new system, configure, install it, and create the Bacula database.
- Ideally, you will have a copy of all the Bacula conf files that were being used on your server. If not, you will at a minimum need create a bacula-dir.conf that has the same Client resource that was used to backup your system.
- If you have a valid saved Bootstrap file as created for your damaged machine with WriteBootstrap, use it to restore the files to the damaged machine, where you have loaded a static Bacula File daemon using the Rescue disk). This is done by using the restore command and at the yes/mod/no prompt, selecting **mod** then specifying the path to the bootstrap file.
- If you have the Bootstrap file, you should now be back up and running, if you do not have a Bootstrap file, continue with the suggestions below.
- Using **bscan** scan the last set of backup tapes into your MySQL, PostgreSQL or SQLite database.
- Start Bacula, and using the Console **restore** command, restore the last valid copy of the Bacula database and the Bacula configuration files.
- Move the database to the correct location.
- Start the database, and restart Bacula. Then use the Console **restore** command, restore all the files on the damaged machine, where you have loaded a Bacula File daemon using the Rescue disk.

For additional details of restoring your database, please see the Restoring When Things Go Wrong section of the Console Restore Command chapter of this manual.

40.9 Linux Problems or Bugs

Since every flavor and every release of Linux is different, there are likely to be some small difficulties with the scripts, so please be prepared to edit them in a minimal environment. A rudimentary knowledge of **vi** is very useful. Also, these scripts do not do everything. You will need to reformat Windows partitions by hand, for example.

Getting the boot loader back can be a problem if you are using **grub** because it is so complicated. If all else fails, reboot your system from your floppy but using the restored disk image, then proceed to a reinstallation of grub (looking at the run-grub script can help). By contrast, lilo is a piece of cake.

40.10 Bare Metal Recovery using a LiveCD

As an alternative to the old now defunct Bacula Rescue CDROM, you can use any system rescue or LiveCD to recover your system. The big problem with most rescue or LiveCDs is that they are not designed to capture the current state of your system, so when you boot them on a damaged system, you might be somewhat lost – e.g. how many of you remember your exact hard disk partitioning.

This lack can be easily corrected by running the part of the Bacula Rescue code that creates a directory containing a static-bacula-fd, a snapshot of your current system disk configuration, and scripts that help restoring it.

Before a disaster strikes:

1. Run only the **make bacula** part of the Bacula Rescue procedure to create the static Bacula File daemon, and system disk snapshot.
2. Save the directory generated (more details below) preferrably on a CDROM or alternatively to some other system.
3. Possibly run **make bacula** every night as part of your backup process to ensure that you have a current snapshot of your system.

Then when disaster strikes, do the following:

1. Boot with your system rescue disk or LiveCD (e.g. Knoppix).
2. Start the Network (local network).
3. Copy the Bacula recovery directory to the damaged system using ftp, scp, wget or if your boot disk permits it reading it directly from a CDROM.
4. Continue as documented above.
5. Re-partition your hard disk(s) as it was before, if necessary.
6. Re-format your partitions, if necessary.
7. Restore the Bacula File daemon (static version).
8. Perform a Bacula restore of all your files.
9. Re-install your boot loader.
10. Reboot.

In order to create the Bacula recovery directory, you need a copy of the Bacula Rescue code as described above, and you must first configure that directory.

Once the configuration is done, you can do the following to create the Bacula recovery directory:

```
cd <bacula-rescue-source>/linux/cdrom
su (become root)
make bacula
```

The directory you want to save will be created in the current directory with the name **bacula**. You need only save that directory either as a directory or possibly as a compressed tar file. If you run this procedure on multiple machines, you will probably want to rename this directory to something like **bacula-hostname**.

40.11 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 first SCSI 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

40.12 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.

40.13 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

40.14 Bugs and Other Considerations

Directory Modification and Access Times are Modified on pre-1.30 Baculas : When a pre-1.30 version of Bacula restores a directory, it first must create the directory, then it populates the directory with its files and subdirectories. The act of creating the files and subdirectories updates both the modification and access times associated with the directory itself. As a consequence, all modification and access times of all directories will be updated to the time of the restore.

This has been corrected in Bacula version 1.30 and later. The directory modification and access times are reset to the value saved in the backup after all the files and subdirectories have been restored. This has been tested and verified on normal restore operations, but not verified during a bare metal recovery.

Strange Bootstrap Files: If any of you look closely at the bootstrap file that is produced and used for the restore (I sure do), you will probably notice that the FileIndex item does not include all the files saved to the tape. This is because in some instances there are duplicates (especially in the case of an Incremental save), and in such circumstances, **Bacula** restores only the last of multiple copies of a file or directory.

40.15 Disaster Recovery of Win32 Systems

Due to open system files, and registry problems, Bacula cannot save and restore a complete Win2K/XP/NT environment.

A suggestion by Damian Coutts using Microsoft's NTBackup utility in conjunction with Bacula should permit a Full bare metal restore of Win2K/XP (and possibly NT systems). 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. This command can be put in a Client Run Before Job directive so that it is automatically run during each backup, and thus saved to a Bacula Volume.

To restore the system state, you first reload a base operating system, then you would use Bacula to restore all the users files and to recover the **c:\systemstate.bkf** file, and finally, 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.

This procedure has been confirmed to work by Ludovic Strappazon – many thanks!

A new tool is provided in the form of a bacula plugin for the BartPE rescue CD. BartPE is a self-contained WindowsXP boot CD which you can make using the PeBuilder tools available at http://www.nu2.nu/pebuilder/ and a valid Windows XP SP1 CDROM. The plugin is provided as a zip archive. Unzip the file and copy the bacula directory into the plugin directory of your BartPE installation. Edit the configuration files to suit your installation and build your CD according to the instructions at Bart's site. This will permit you to boot from the cd, configure and start networking, start the bacula file client and access your director with the console program. The programs menu on the booted CD contains entries to install the file client service, start the file client service, and start the WX-Console. You can also open a command line window and CD Programs\Bacula and run the command line console bconsole.

40.16 Ownership and Permissions on Win32 Systems

Bacula versions after 1.31 should properly restore ownership and permissions on all WinNT/XP/2K systems. If you do experience problems, generally in restores to alternate directories because higher level directories were not backed up by Bacula, you can correct any problems with the **SetACL** available under the GPL license at: http://sourceforge.net/projects/setacl/.

40.17 Alternate Disaster Recovery Suggestion for Win32 Systems

Ludovic Strappazon has suggested an interesting way to backup and restore complete Win32 partitions. Simply boot your Win32 system with a Linux Rescue disk as described above for Linux, install a statically linked Bacula, and backup any of the raw partitions you want. Then to restore the system, you simply restore the raw partition or partitions. Here is the email that Ludovic recently sent on that subject:

```
I've just finished testing my brand new cd LFS/Bacula
with a raw Bacula backup and restore of my portable.
I can't resist sending you the results: look at the rates !!!
hunt-dir: Start Backup JobId 100, Job=HuntBackup.2003-04-17_12.58.26
hunt-dir: Bacula 1.30 (14Apr03): 17-Apr-2003 13:14
JobId:                  100
Job:                    HuntBackup.2003-04-17_12.58.26
FileSet:                RawPartition
Backup Level:           Full
Client:                 sauvegarde-fd
Start time:             17-Apr-2003 12:58
End time:               17-Apr-2003 13:14
Files Written:          1
Bytes Written:          10,058,586,272
Rate:                   10734.9 KB/s
Software Compression:   None
Volume names(s):        000103
Volume Session Id:      2
Volume Session Time:    1050576790
Last Volume Bytes:      10,080,883,520
FD termination status:  OK
SD termination status:  OK
Termination:            Backup OK
hunt-dir: Begin pruning Jobs.
hunt-dir: No Jobs found to prune.
hunt-dir: Begin pruning Files.
hunt-dir: No Files found to prune.
hunt-dir: End auto prune.
hunt-dir: Start Restore Job RestoreFilesHunt.2003-04-17_13.21.44
hunt-sd: Forward spacing to file 1.
hunt-dir: Bacula 1.30 (14Apr03): 17-Apr-2003 13:54
JobId:                  101
Job:                    RestoreFilesHunt.2003-04-17_13.21.44
Client:                 sauvegarde-fd
Start time:             17-Apr-2003 13:21
```

```
End time:              17-Apr-2003 13:54
Files Restored:        1
Bytes Restored:        10,056,130,560
Rate:                  5073.7 KB/s
FD termination status:  OK
Termination:           Restore OK
hunt-dir: Begin pruning Jobs.
hunt-dir: No Jobs found to prune.
hunt-dir: Begin pruning Files.
hunt-dir: No Files found to prune.
hunt-dir: End auto prune.
```

40.18 Restoring to a Running System

If for some reason you want to do a Full restore to a system that has a working kernel (not recommended), you will need to take care not to overwrite the following files:

```
/etc/grub.conf
/etc/X11/Conf
/etc/fstab
/etc/mtab
/lib/modules
/usr/modules
/usr/X11R6
/etc/modules.conf
```

40.19 Additional Resources

Many thanks to Charles Curley who wrote Linux Complete Backup and Recovery HOWTO for the The Linux Documentation Project. This is an excellent document on how to do Bare Metal Recovery on Linux systems, and it was this document that made me realize that Bacula could do the same thing.

You can find quite a few additional resources, both commercial and free at Storage Mountain, formerly known as Backup Central.

And finally, the O'Reilly book, "Unix Backup & Recovery" by W. Curtis Preston covers virtually every backup and recovery topic including bare metal recovery for a large range of Unix systems.

Chapter 41

Bacula TLS – Communications Encryption

Bacula TLS (Transport Layer Security) is built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh**. The data written to Volumes by the Storage daemon is not encrypted by this code. For data encryption, please see the Data Encryption Chapter of this manual.

The Bacula encryption implementations were written by Landon Fuller.

Supported features of this code include:

- Client/Server TLS Requirement Negotiation
- TLSv1 Connections with Server and Client Certificate Validation
- Forward Secrecy Support via Diffie-Hellman Ephemeral Keying

This document will refer to both "server" and "client" contexts. These terms refer to the accepting and initiating peer, respectively.

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 if enabled on the **./configure** line with **--with-openssl**

41.1 TLS Configuration Directives

Additional configuration directives have been added to all the daemons (Director, File daemon, and Storage daemon) as well as the various different Console programs. 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 Require = <yes—no> Require TLS connections. This directive is ignored unless **TLS Enable** is set to **yes**. If TLS is not required, and TLS is enabled, then Bacula will connect with other daemons either with or without TLS depending on what the other daemon requests. If TLS is enabled and TLS is required, then Bacula will refuse any connection that does not use TLS.

TLS Certificate = <Filename> The full path and filename of a PEM encoded TLS certificate. It can be used as either a client or server certificate. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. It is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

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 x509 certificate. Any client certificate signed by a known-CA will be accepted unless the TLS Allowed CN configuration directive is used, in which case the client certificate must correspond to the Allowed Common Name specified. This directive is valid only for a server and not in a client context.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. If this directive is specified, all server certificates will be verified against this list. This can be used to ensure that only the CA-approved Director may connect. This directive may be specified more than once.

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 if *TLS Verify*

Peer (see above) is also specified, 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 bf .0. One of *TLS CA Certificate File* or *TLS CA Certificate Dir* are required in a server context if *TLS Verify Peer* is also specified, 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 dh1024.pem -5 1024
```

41.2 Creating a Self-signed Certificate

You may create a self-signed certificate for use with the Bacula TLS that will permit you to make it function, but will not allow certificate validation. The .pem file containing both the certificate and the key valid for ten years can be made with the following:

```
openssl req -new -x509 -nodes -out bacula.pem -keyout bacula.pem -days 3650
```

The above script will ask you 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. Essentially, if the Director does not verify that it is talking to a trusted remote endpoint, it can be tricked into talking to a malicious 3rd 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.

To get a trusted certificate (CA or Certificate Authority signed certificate), you will either need to purchase certificates signed by a commercial CA or find a friend that has setup his own CA or 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 explains how to do it, or you can read the documentation provided in the Open-source PKI Book project at Source Forge: http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm. Note, this link may change.

The program TinyCA has a very nice Graphical User Interface that allows you to easily setup and maintain your own CA. TinyCA can be found at http://tinyca.sm-zone.net/.

41.3 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 quite 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 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 explains how to do it, or you can read the documentation provided in the Open-source PKI Book project at Source Forge: http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm. Note, this link may change.

41.4 Example TLS Configuration Files

Landon has supplied us with the TLS portions of his configuration files, which should help you setting up your own. Note, this example shows the directives necessary for a Director to Storage daemon session. The technique is the same between the Director and the Client and for bconsole to the Director.

bacula-dir.conf

```
Director {                            # define myself
  Name = backup1-dir
  ...
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  TLS Allowed CN = "bacula@backup1.example.com"
  TLS Allowed CN = "administrator@example.com"
  TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
  # This is a server certificate, used for incoming
  # console connections.
  TLS Certificate = /usr/local/etc/ssl/backup1/cert.pem
  TLS Key = /usr/local/etc/ssl/backup1/key.pem
}

Storage {
  Name = File
  Address = backup1.example.com
  ...
  TLS Require = yes
  TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
  # This is a client certificate, used by the director to
  # connect to the storage daemon
  TLS Certificate = /usr/local/etc/ssl/bacula@backup1/cert.pem
  TLS Key = /usr/local/etc/ssl/bacula@backup1/key.pem
}

Client {
  Name = backup1-fd
  Address = server1.example.com
  ...

  TLS Enable = yes
  TLS Require = yes
  TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
}
```

bacula-fd.conf

```
Director {
  Name = backup1-dir
  ...
  TLS Enable = yes
  TLS Require = yes
  TLS Verify Peer = yes
  # Allow only the Director to connect
```

```
      TLS Allowed CN = "bacula@backup1.example.com"
      TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
      # This is a server certificate. It is used by connecting
      # directors to verify the authenticity of this file daemon
      TLS Certificate = /usr/local/etc/ssl/server1/cert.pem
      TLS Key = /usr/local/etc/ssl/server1/key.pem
    }

   FileDaemon {
      Name = backup1-fd
      ...
      # you need these TLS entries so the SD and FD can
      # communicate
      TLS Enable = yes
      TLS Require = yes

      TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
      TLS Certificate = /usr/local/etc/ssl/server1/cert.pem
      TLS Key = /usr/local/etc/ssl/server1/key.pem
}
```

bacula-sd.conf

```
    Storage {                             # definition of myself
      Name = backup1-sd
      ...
      # These TLS configuration options are used for incoming
      # file daemon connections. Director TLS settings are handled
      # below.
      TLS Enable = yes
      TLS Require = yes
      # Peer certificate is not required/requested -- peer validity
      # is verified by the storage connection cookie provided to the
      # File Daemon by the director.
      TLS Verify Peer = no
      TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
      # This is a server certificate. It is used by connecting
      # file daemons to verify the authenticity of this storage daemon
      TLS Certificate = /usr/local/etc/ssl/backup1/cert.pem
      TLS Key = /usr/local/etc/ssl/backup1/key.pem
    }

    #
    # List Directors who are permitted to contact Storage daemon
    #
    Director {
      Name = backup1-dir
      ...
      TLS Enable = yes
      TLS Require = yes
      # Require the connecting director to provide a certificate
      # with the matching CN.
      TLS Verify Peer = yes
```

```
  TLS Allowed CN = "bacula@backup1.example.com"
  TLS CA Certificate File = /usr/local/etc/ssl/ca.pem
  # This is a server certificate. It is used by the connecting
  # director to verify the authenticity of this storage daemon
  TLS Certificate = /usr/local/etc/ssl/backup1/cert.pem
  TLS Key = /usr/local/etc/ssl/backup1/key.pem
}
```

Chapter 42

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 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 OS X resource forks are encrypted.

Encryption and signing are implemented using RSA private keys coupled with self-signed x509 public certificates. This is also sometimes known as PKI or Public Key Infrastructure.

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.

42.1 Building Bacula with Encryption Support

The configuration option for enabling OpenSSL encryption support has not changed since Bacula 1.38. To build Bacula with encryption support, you will need the OpenSSL libraries and headers installed. When configuring Bacula, use:

```
./configure --with-openssl ...
```

42.2 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.

42.3 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.

You must:

- Concatenate the master private and public key into a single keypair file, ie: cat master.key master.cert ¿master.keypair
- 2) Set the PKI Keypair statement in your bacula configuration file:

  ```
  PKI Keypair = master.keypair
  ```

- Start the restore. The master keypair will be used to decrypt the file data.

42.4 Generating Private/Public Encryption Keys

Generate a Master Key Pair with:

```
openssl genrsa -out master.key 2048
openssl req -new -key master.key -x509 -out master.cert
```

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
```

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 (x509) 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 X509 certificate encoding that contains only a single public key.

42.5 Example Data Encryption Configuration

bacula-fd.conf

```
FileDaemon {
   Name = example-fd
   FDport = 9102                  # where we listen for the director
   WorkingDirectory = /var/bacula/working
   Pid Directory = /var/run
   Maximum Concurrent Jobs = 20

   PKI Signatures = Yes            # Enable Data Signing
   PKI Encryption = Yes            # Enable Data Encryption
   PKI Keypair = "/etc/bacula/fd-example.pem"    # Public and Private Keys
   PKI Master Key = "/etc/bacula/master.cert"    # ONLY the Public Key
}
```

Chapter 43

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 (transport layer security) 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.

43.1 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 100should:

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.

43.2 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 tcpdchk(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
```

43.3 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 44

Dealing with Firewalls

If you have a firewall or a DMZ installed on your computer, you may experience difficulties contacting one or more of the Clients to back them up. This is especially true if you are trying to backup a Client across the Internet.

44.1 Technical Details

If you are attempting to do this, the sequence of network events in Bacula to do a backup are the following:

```
Console -> DIR:9101
DIR     -> SD:9103
DIR     -> FD:9102
FD      -> SD:9103
```

Where hopefully it is obvious that DIR represents the Director, FD the File daemon or client, and SD the Storage daemon. The numbers that follow those names are the standard ports used by Bacula, and the -> represents the left side making a connection to the right side (i.e. the right side is the "server" or is listening on the specified port), and the left side is the "client" that initiates the conversation.

Note, port 9103 serves both the Director and the File daemon, each having its own independent connection.

If you are running **iptables**, you might add something like:

```
-A FW-1-INPUT -m state --state NEW -m tcp -p tcp --dport 9101:9103 -j ACCEPT
```

on your server, and

```
-A FW-1-INPUT -m state --state NEW -m tcp -p tcp --dport 9102 -j ACCEPT
```

on your client. In both cases, I assume that the machine is allowed to initiate connections on any port. If not, you will need to allow outgoing connections on ports 9102 and 9103 on your server and 9103 on your client. Thanks to Raymond Norton for this tip.

44.2 A Concrete Example

The following discussion was originally written by Jesse Guardiani because he has 'internal' and 'external' requiring the Director and the Client to use different IP addresses. His original solution was to define two different Storage resources in the Director's conf file each pointing to the same Storage daemon but with different IP addresses. In Bacula 1.38.x this no longer works, because Bacula makes a one-to-one association between a Storage daemon resource and a Device (such as an Autochanger). As a consequence, I have modified his original text to a method that I believe will work, but is as of yet untested (KES - July 2006).

My bacula server is on the 192.168.1.0/24 network at IP address 192.168.1.52. For the sake of discussion we will refer to this network as the 'internal' network because it connects to the internet through a NAT'd firewall. We will call the network on the public (internet) side of the NAT'd firewall the 'external' network. Also, for the sake of discussion we will call my bacula server:

```
    server.int.mydomain.tld
```

when a fully qualified domain name is required, or simply:

```
    server
```

if a hostname is adequate. We will call the various bacula daemons running on the server.int.mydomain.tld machine:

```
    server-fd
    server-sd
    server-dir
```

In addition, I have two clients that I want to back up with Bacula. The first client is on the internal network. Its fully qualified domain name is:

```
private1.int.mydomain.tld
```

And its hostname is:

```
private1
```

This machine is a client and therefore runs just one bacula daemon:

```
private1-fd
```

The second client is on the external network. Its fully qualified domain name is:

```
public1.mydomain.tld
```

And its hostname is:

```
public1
```

This machine also runs just one bacula daemon:

```
public1-fd
```

Finally, I have a NAT firewall/gateway with two network interfaces. The first interface is on the internal network and serves as a gateway to the internet for all the machines attached to the internal network (For example, server.int.mydomain.tld and private1.int.mydomain.tld). The second interface is on the external (internet) network. The external interface has been assigned the name:

```
firewall.mydomain.tld
```

Remember:

```
*.int.mydomain.tld = internal network
    *.mydomain.tld = external network
```

44.2.1 The Bacula Configuration Files for the Above

server-sd manages a 4 tape AIT autoloader. All of my backups are written to server-sd. I have just *one* Device resource in my server-sd.conf file:

```
Autochanger {
  Name = "autochanger1";\
  Device = Drive0
  Changer Device = /dev/ch0;
  Changer Command = "/usr/local/sbin/chio-bacula %c %o %S %a";
}
Device {
  Name = Drive0
  DriveIndex = 0
  Media Type = AIT-1;
  Archive Device = /dev/nrsa1;
  Label Media = yes;
  AutoChanger = yes;
  AutomaticMount = yes;              # when device opened, read it
  AlwaysOpen = yes;
  Hardware End of Medium = No
  Fast Forward Space File = No
  BSF at EOM = yes
}
```

(note, please see the Tape Testing chapter of this manual for important FreeBSD information.) However, unlike previously, there is only one Storage definition in my server-dir.conf file:

```
Storage {
  Name = "autochanger1"    # Storage device for backing up
  Address = Storage-server
  SDPort = 9103
  Password = "mysecretpassword"
  Device = "autochanger1"
  Media Type = AIT-1
  Autochanger = yes
}
```

Note that the Storage resource uses neither of the two addresses to the Storage daemon – neither server.int.mydomain.tld nor firewall.mydomain.tld, but instead uses the address Storage-server.

What is key is that in the internal net, Storage-server is resolved to server.int.mydomain.tld, either with an entry in /etc/hosts, or by creating and appropriate DNS entry, and on the external net (the Client machine), Storage-server is resolved to firewall.mydomain.tld.

In addition to the above, I have two Client resources defined in server-dir.conf:

```
Client {
  Name = private1-fd
  Address = private1.int.mydomain.tld
  FDPort = 9102
  Catalog = MyCatalog
  Password = "mysecretpassword"       # password for FileDaemon
}
Client {
  Name = public1-fd
  Address = public1.mydomain.tld
  FDPort = 9102
  Catalog = MyCatalog
  Password = "mysecretpassword"       # password for FileDaemon
}
```

And finally, to tie it all together, I have two Job resources defined in server-dir.conf:

```
Job {
  Name = "Private1-Backup"
  Type = Backup
  Client = private1-fd
  FileSet = "Private1"
  Schedule = "WeeklyCycle"
  Storage = "autochanger1-int"
  Messages = Standard
  Pool = "Weekly"
  Write Bootstrap = "/var/db/bacula/Private1-Backup.bsr"
  Priority = 12
}
Job {
  Name = "Public1-Backup"
  Type = Backup
  Client = public1-fd
  FileSet = "Public1"
  Schedule = "WeeklyCycle"
  Storage = "autochanger1-ext"
  Messages = Standard
  Pool = "Weekly"
  Write Bootstrap = "/var/db/bacula/Public1-Backup.bsr"
  Priority = 13
}
```

It is important to notice that because the 'Private1-Backup' Job is intended to back up a machine on the internal network so it resolves Storage-server to contact the Storage daemon via the internal net. On the other hand, the 'Public1-Backup' Job is intended to back up a machine on the external network, so it resolves Storage-server to contact the Storage daemon via the external net.

I have left the Pool, Catalog, Messages, FileSet, Schedule, and Director resources out of the above server-dir.conf examples because they are not pertinent to the discussion.

44.2.2 How Does It Work?

If I want to run a backup of private1.int.mydomain.tld and store that backup using server-sd then my understanding of the order of events is this:

1. I execute my Bacula 'console' command on server.int.mydomain.tld.
2. console connects to server-dir.
3. I tell console to 'run' backup Job 'Private1-Backup'.
4. console relays this command to server-dir.
5. server-dir connects to private1-fd at private1.int.mydomain.tld:9102
6. server-dir tells private1-fd to start sending the files defined in the 'Private1-Backup' Job's FileSet resource to the Storage resource 'autochanger1', which we have defined in server-dir.conf as having the address:port of Storage-server, which is mapped by DNS to server.int.mydomain.tld.
7. private1-fd connects to server.int.mydomain.tld:9103 and begins sending files.

Alternatively, if I want to run a backup of public1.mydomain.tld and store that backup using server-sd then my understanding of the order of events is this:

1. I execute my Bacula 'console' command on server.int.mydomain.tld.
2. console connects to server-dir.
3. I tell console to 'run' backup Job 'Public1-Backup'.
4. console relays this command to server-dir.

5. server-dir connects, through the NAT'd firewall, to public1-fd at public1.mydomain.tld:9102

6. server-dir tells public1-fd to start sending the files defined in the 'Public1-Backup' Job's FileSet resource to the Storage resource 'autochanger1', which we have defined in server-dir.conf as having the same address:port as above of Storage-server, but which on this machine is resolved to firewall.mydomain.tld:9103.

7. public1-fd connects to firewall.mydomain.tld:9103 and begins sending files.

44.2.3 Important Note

In order for the above 'Public1-Backup' Job to succeed, firewall.mydomain.tld:9103 MUST be forwarded using the firewall's configuration software to server.int.mydomain.tld:9103. Some firewalls call this 'Server Publication'. Others may call it 'Port Forwarding'.

44.2.4 Firewall Problems

Either a firewall or a router may decide to timeout and terminate open connections if they are not active for a short time. By Internet standards the period should be two hours, and should be indefinitely extended if KEEPALIVE is set as is the case by Bacula. If your firewall or router does not respect these rules, you may find Bacula connections terminated. In that case, the first thing to try is turning on the **Heart Beat Interval** both in the File daemon and the Storage daemon and set an interval of say five minutes.

Also, if you have denial of service rate limiting in your firewall, this too can cause Bacula disconnects since Bacula can at times use very high access rates. To avoid this, you should implement default accept rules for the Bacula ports involved before the rate limiting rules.

Finally, if you have a Windows machine, it will most likely by default disallow connections to the Bacula Windows File daemon. See the Windows chapter of this manual for additional details.

Chapter 45

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.

45.1 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.

45.2 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
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: Job
     2: Level
     3: FileSet
     4: Client
     5: Storage
Select parameter to modify (1-5): 2
```

you should select number 2 to modify the **Level**, and it will display:

```
Levels:
     1: Initialize Catalog
     2: Verify from Catalog
     3: Verify Volume
     4: 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
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.

45.3 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.

45.4 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 = { }
P
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 46

Bacula RPM Packaging FAQ

1. How do I build Bacula for platform xxx?
2. How do I control which database support gets built?
3. What other defines are used?
4. I'm getting errors about not having permission when I try to build the packages. Do I need to be r
5. I'm building my own rpms but on all platforms and compiles I get an unresolved dependency for so
6. I'm building my own rpms because you don't publish for my platform. Can I get my packages relea
7. Is there an easier way than sorting out all these command line options?
8. I just upgraded from 1.36.x to 1.38.x and now my director daemon won't start. It appears to start
9. There are a lot of rpm packages. Which packages do I need for what?

46.1 Answers

1. **How do I build Bacula for platform xxx?** The bacula spec file contains defines to build for several platforms: Red Hat 7.x (rh7), Red Hat 8.0 (rh8), Red Hat 9 (rh9), Fedora Core (fc1, fc3, fc4, fc5, fc6, fc7, fc8), Whitebox Enterprise Linux 3.0 (wb3), Red Hat Enterprise Linux (rhel3, rhel4, rhel5), Mandrake 10.x (mdk), Mandriva 2006.x (mdv) CentOS (centos3, centos4, centos5) Scientific Linux (sl3, sl4, sl5) and SuSE (su9, su10, su102, su103, su110). The package build is controlled by a mandatory define set at the beginning of the file. These defines basically just control the dependency information that gets

coded into the finished rpm package as well as any special configure options required. The platform define may be edited in the spec file directly (by default all defines are set to 0 or "not set"). For example, to build the Red Hat 7.x package find the line in the spec file which reads

```
%define rh7 0
```

and edit it to read

```
%define rh7 1
```

Alternately you may pass the define on the command line when calling rpmbuild:

```
rpmbuild -ba --define "build_rh7 1" bacula.spec
rpmbuild --rebuild --define build_rh7 1" bacula-x.x.x-x.src.rpm
```

2. **How do I control which database support gets built?** Another mandatory build define controls which database support is compiled, one of build_sqlite, build_mysql or build_postgresql. To get the MySQL package and support either set the

```
%define mysql 0
OR
%define mysql4 0
OR
%define mysql5 0
```

to

```
%define mysql 1
OR
%define mysql4 1
OR
%define mysql5 1
```

in the spec file directly or pass it to rpmbuild on the command line:

```
rpmbuild -ba --define "build_rh7 1" --define "build_mysql 1" bacula.spec
rpmbuild -ba --define "build_rh7 1" --define "build_mysql4 1" bacula.spec
rpmbuild -ba --define "build_rh7 1" --define "build_mysql5 1" bacula.spec
```

3. **What other defines are used?** Three other building defines of note are the depkgs_version, docs_version and _rescuever identifiers. These two defines are set with each release and must match the version of those sources that are being used to build the packages. You would not ordinarily need to edit these. See also the Build Options section below for other build time options that can be passed on the command line.

4. **I'm getting errors about not having permission when I try to build the packages. Do I need to be root?** No, you do not need to be root and, in fact, it is better practice to build rpm packages as a non-root user. Bacula packages are designed to be built by a regular user but you must make a few changes on your system to do this. If you are building on your own system then the simplest method is to add write permissions for all to the build directory (/usr/src/redhat/, /usr/src/RPM or /usr/src/packages). To accomplish this, execute the following command as root:

```
chmod -R 777 /usr/src/redhat
chmod -R 777 /usr/src/RPM
chmod -R 777 /usr/src/packages
```

   If you are working on a shared system where you can not use the method above then you need to recreate the appropriate above directory tree with all of its subdirectories inside your home directory. Then create a file named

   `.rpmmacros`

   in your home directory (or edit the file if it already exists) and add the following line:

```
%_topdir /home/myuser/redhat
```

   Another handy directive for the .rpmmacros file if you wish to suppress the creation of debug rpm packages is:

```
%debug_package %{nil}
```

5. **I'm building my own rpms but on all platforms and compiles I get an unresolved dependency for something called /usr/afsws/bin/pagsh.** This is a shell from the OpenAFS (Andrew File System). If you are seeing this then you chose to include the

docs/examples directory in your package. One of the example scripts in this directory is a pagsh script. Rpmbuild, when scanning for dependencies, looks at the shebang line of all packaged scripts in addition to checking shared libraries. To avoid this do not package the examples directory. If you are seeing this problem you are building a very old bacula package as the examples have been removed from the doc packaging.

6. **I'm building my own rpms because you don't publish for my platform. Can I get my packages released to sourceforge for other people to use?** Yes, contributions from users are accepted and appreciated. Please examine the directory platforms/contrib-rpm in the source code for further information.

7. **Is there an easier way than sorting out all these command line options?** Yes, there is a gui wizard shell script which you can use to rebuild the src rpm package. Look in the source archive for platforms/contrib-rpm/rpm_wizard.sh. This script will allow you to specify build options using GNOME dialog screens. It requires zenity.

8. **I just upgraded from 1.36.x to 1.38.x and now my director daemon won't start. It appears to start but dies silently and I get a "connection refused" error when starting the console. What is wrong?** Beginning with 1.38 the rpm packages are configured to run the director and storage daemons as a non-root user. The file daemon runs as user root and group bacula, the storage daemon as user bacula and group disk, and the director as user bacula and group bacula. If you are upgrading you will need to change some file permissions for things to work. Execute the following commands as root:

```
chown bacula.bacula /var/bacula/*
chown root.bacula /var/bacula/bacula-fd.9102.state
chown bacula.disk /var/bacula/bacula-sd.9103.state
```

   Further, if you are using File storage volumes rather than tapes those files will also need to have ownership set to user bacula and group bacula.

9. **There are a lot of rpm packages. Which packages do I need for what?** For a bacula server you need to select the packsge based upon your preferred catalog database: one of bacula-mysql, bacula-postgresql or bacula-sqlite. If your system does not provide an mtx package you also need bacula-mtx to satisfy that dependancy. For a client machine you need only install bacula-client. Optionally, for

either server or client machines, you may install a graphical console bacula-gconsole and/or bacula-wxconsole. The Bacula Administration Tool is installed with the bacula-bat package. One last package, bacula-updatedb is required only when upgrading a server more than one database revision level.

10. **Support for RHEL3/4/5, CentOS 3/4/5, Scientific Linux 3/4/5 and x86_64** The examples below show explicit build support for RHEL4 and CentOS 4. Build support for x86_64 has also been added.

```
Build with one of these 3 commands:

rpmbuild --rebuild \
        --define "build_rhel4 1" \
        --define "build_sqlite 1" \
        bacula-1.38.3-1.src.rpm

rpmbuild --rebuild \
        --define "build_rhel4 1" \
        --define "build_postgresql 1" \
        bacula-1.38.3-1.src.rpm

rpmbuild --rebuild \
        --define "build_rhel4 1" \
        --define "build_mysql4 1" \
        bacula-1.38.3-1.src.rpm

For CentOS substitute '--define "build_centos4 1"' in place of rhel4.
For Scientific Linux substitute '--define "build_sl4 1"' in place of rhel4.

For 64 bit support add '--define "build_x86_64 1"'
```

46.2 Build Options

The spec file currently supports building on the following platforms:

```
Red Hat builds
--define "build_rh7 1"
--define "build_rh8 1"
--define "build_rh9 1"

Fedora Core build
--define "build_fc1 1"
--define "build_fc3 1"
--define "build_fc4 1"
--define "build_fc5 1"
--define "build_fc6 1"
```

```
--define "build_fc7 1"
--define "build_fc8 1"
--define "build_fc9 1"

Whitebox Enterprise build
--define "build_wb3 1"

Red Hat Enterprise builds
--define "build_rhel3 1"
--define "build_rhel4 1"
--define "build_rhel5 1"

CentOS build
--define "build_centos3 1"
--define "build_centos4 1"
--define "build_centos5 1"

Scientific Linux build
--define "build_sl3 1"
--define "build_sl4 1"
--define "build_sl5 1"

SuSE build
--define "build_su9 1"
--define "build_su10 1"
--define "build_su102 1"
--define "build_su103 1"
--define "build_su110 1"
--define "build_su111 1"

Mandrake 10.x build
--define "build_mdk 1"

Mandriva build
--define "build_mdv 1"

MySQL support:
for mysql 3.23.x support define this
--define "build_mysql 1"
if using mysql 4.x define this,
currently: Mandrake 10.x, Mandriva 2006.0, SuSE 9.x & 10.0, FC4 & RHEL4
--define "build_mysql4 1"
if using mysql 5.x define this,
currently: SuSE 10.1 & FC5
--define "build_mysql5 1"

PostgreSQL support:
--define "build_postgresql 1"

Sqlite support:
--define "build_sqlite 1"

Build the client rpm only in place of one of the above database full builds:
--define "build_client_only 1"
```

```
X86-64 support:
--define "build_x86_64 1"

Supress build of bgnome-console:
--define "nobuild_gconsole 1"

Build the WXWindows console:
requires wxGTK >= 2.6
--define "build_wxconsole 1"

Build the Bacula Administration Tool:
requires QT >= 4.2
--define "build_bat 1"

Build python scripting support:
--define "build_python 1"

Modify the Packager tag for third party packages:
--define "contrib_packager Your Name <youremail@site.org>"

Install most files to /opt/bacula directory:
--define "single_dir_install 1"

Supress building the rescue files:
--define "nobuild_rescue 1"
```

46.3 RPM Install Problems

In general the RPMs, once properly built should install correctly. However, when attempting to run the daemons, a number of problems can occur:

/var/bacula Permissions By default, the Director and Storage daemon do not run with root permission. If the /var/bacula is owned by root, then it is possible that the Director and the Storage daemon will not be able to access this directory, which is used as the Working Directory. To fix this, the easiest thing to do is:

```
chown bacula:bacula /var/bacula
```

Note: as of 1.38.8 /var/bacula is installed root:bacula with permissions 770.

ot Access the Tape drive This can happen in some older RPM releases where the Storage daemon ran under userid bacula, group bacula. There are two ways of fixing this: the best is to modify the /etc/init.d/bacula-sd file so that it starts the Storage daemon with group "disk". The second way to fix the problem is to change the permissions of your tape drive (usually /dev/nst0) so that Bacula can access it. You will probably need to change the permissions of the SCSI control device as well, which is usually /dev/sg0. The exact names depend on your configuration, please see the Tape Testing chapter for more information on devices.

Chapter 47

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 restore_command command in the Console program, 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.

47.1 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.

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
```

47.2 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.

47.3 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"
```

47.4 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.

Chapter 48

Installing and Configuring MySQL

48.1 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
```

The names of the packages may vary from distribution to distribution. It is important to have the devel 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 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.

48.2 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;
```

48.3 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.

48.4 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>
```

48.5 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.

48.6 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.

Chapter 49

Installing and Configuring PostgreSQL

If you are considering using PostreSQL, 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.

49.1 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.

If you are using FreeBSD, this FreeBSD Diary article 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:

```
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, ...).

49.2 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.

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_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, ...). Ideally, the encoding should be set to UTF8. However, many Unix systems have filenames that are not encoded in UTF8, either because you have not set UTF8 as your default character set or because you have imported files from elsewhere (e.g. MacOS X). For this reason, Bacula uses SQL_ASCII as the default encoding. If you want to change this, please modify the script before running it, but be forewarned that Bacula backups will fail if PostgreSQL finds any non-UTF8 sequences.

   If running the script fails, it is probably because the database is owned by a user other than yourself. On many systems, the database 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 (or postgres)
createuser kern (or perhaps 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
```

At this point, you should be able to execute the ./create_bacula_database command.

3. ./make_bacula_tables

   This script creates the PostgreSQL tables used by **Bacula**.

4. ./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
```

Also, I had an authorization problem with the password. In the end, I had to modify my **pg_hba.conf** file (in /var/lib/pgsql/data on my machine) from:

```
  local   all    all        ident  sameuser
to
  local   all    all        trust
```

This solved the problem for me, but it is not always a good thing to do from a security standpoint. However, it allowed me 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 just prior the the existing "local" and "host" lines, add the line:

```
local bacula bacula md5
```

and restart the Postgres 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.

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 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 for details.
RunBeforeJob = "/etc/make_catalog_backup bacula bacula secret"
```

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.

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 Postgres is available to all users of the local system. To eliminate this problem, the Postgres 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.

49.3 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.

49.4 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 package, so you must install:

```
postgresql
postgresql-devel
postgresql-server
postgresql-libs
```

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.

49.5 Converting from MySQL to PostgreSQL

The conversion procedure presented here was worked out by Norm Dressler <ndressler at dinmar dot com>

This process was tested using the following software versions:

- Linux Mandrake 10/Kernel 2.4.22-10 SMP
- Mysql Ver 12.21 Distrib 4.0.15, for mandrake-linux-gnu (i586)
- PostgreSQL 7.3.4
- Bacula 1.34.5

WARNING: Always as a precaution, take a complete backup of your databases before proceeding with this process!

1. Shutdown bacula (cd /etc/bacula;./bacula stop)
2. Run the following command to dump your Mysql database:

   ```
   mysqldump -f -t -n >bacula-backup.dmp
   ```

3. Make a backup of your /etc/bacula directory (but leave the original in place).
4. Go to your Bacula source directory and rebuild it to include PostgreSQL support rather then Mysql support. Check the config.log file for your original configure command and replace enable-mysql with enable-postgresql.
5. Recompile Bacula with a make and if everything compiles completely, perform a make install.
6. Shutdown Mysql.
7. Start PostgreSQL on your system.

8. Create a bacula user in Postgres with the createuser command. Depending on your Postgres install, you may have to SU to the user who has privileges to create a user.

9. Verify your pg_hba.conf file contains sufficient permissions to allow bacula to access the server. Mine has the following since it's on a secure network:

```
local all all trust

host all all 127.0.0.1 255.255.255.255 trust

NOTE: you should restart your postgres server if you
      made changes
```

10. Change into the /etc/bacula directory and prepare the database and tables with the following commands:

```
./create_postgresql_database

./make_postgresql_tables

./grant_postgresql_privileges
```

11. Verify you have access to the database:

```
psql -Ubacula bacula
```

    You should not get any errors.

12. Load your database from the Mysql database dump with:

```
psql -Ubacula bacula <bacula-backup.dmp>
```

13. Resequence your tables with the following commands:

```
psql -Ubacula bacula

SELECT SETVAL('basefiles_baseid_seq', (SELECT
MAX(baseid) FROM basefiles));
SELECT SETVAL('client_clientid_seq', (SELECT
MAX(clientid) FROM client));
SELECT SETVAL('file_fileid_seq', (SELECT MAX(fileid)
```

```
FROM file));
SELECT SETVAL('filename_filenameid_seq', (SELECT
MAX(filenameid) FROM filename));

SELECT SETVAL('fileset_filesetid_seq', (SELECT
MAX(filesetid) FROM fileset));

SELECT SETVAL('job_jobid_seq', (SELECT MAX(jobid) FROM job));
SELECT SETVAL('jobmedia_jobmediaid_seq', (SELECT
MAX(jobmediaid) FROM jobmedia));
SELECT SETVAL('media_mediaid_seq', (SELECT MAX(mediaid) FROM media));
SELECT SETVAL('path_pathid_seq', (SELECT MAX(pathid) FROM path));

SELECT SETVAL('pool_poolid_seq', (SELECT MAX(poolid) FROM pool));
```

14. At this point, start up Bacula, verify your volume library and perform a test backup to make sure everything is working properly.

49.6 Upgrading PostgreSQL

If you upgrade PostgreSQL, you must reconfigure, rebuild, and re-install Bacula otherwise you are likely to get bizarre failures. If you to modify the bacula.spec file to account for the new PostgreSQL version. You can do so by rebuilding from the source rpm. To do so, you may need install from rpms and you upgrade PostgreSQL, you must also rebuild Bacula.

49.7 Credits

Many thanks to Dan Langille for writing the PostgreSQL driver. This will surely become the most popular database that Bacula supports.

Chapter 50

Installing and Configuring SQLite

Please note that SQLite both versions 2 and 3 are not network enabled, which means that they must be linked into the Director rather than accessed by the network as MySQL and PostgreSQL are. This has two consequences:

1. SQLite cannot be used in the **bweb** web GUI package.
2. If you use SQLite, and your Storage daemon is not on the same machine as your Director, you will need to transfer your database to the Storage daemon's machine before you can use any of the SD tools such as **bscan**, ...

50.1 Installing and Configuring SQLite – Phase I

If you use the **./configure --with-sqlite** statement for configuring **Bacula**, you will need SQLite version 2.8.16 or later installed. Our standard location (for the moment) for SQLite is in the dependency package **depkgs/sqlite-2.8.16**. Please note that the version will be updated as new versions are available and tested.

Installing and Configuring is quite easy.

1. Download the Bacula dependency packages
2. Detar it with something like:

tar xvfz depkgs.tar.gz

Note, the above command requires GNU tar. If you do not have GNU tar, a command such as:

zcat depkgs.tar.gz — tar xvf -

will probably accomplish the same thing.

3. **cd depkgs**

4. **make sqlite**

Please note that the **./configure** used to build **Bacula** will need to include **--with-sqlite** or **--with-sqlite3** depending one which version of SQLite you are using. You should not use the **--enable-batch-insert** configuration parameter for Bacula if you are using SQLite version 2 as it is probably not thread safe. If you are using SQLite version 3, you may use the **--enable-batch-insert** configuration option with Bacula, but when building SQLite3 you MUST configure it with **--enable-threadsafe** and **--enable-cross-thread-connections**.

By default, SQLite3 is now run with **PRAGMA synchronous=OFF** this increases the speed by more than 30 time, but it also increases the possibility of a corrupted database if your server crashes (power failure or kernel bug). If you want more security, you can change the PRAGMA that is used in the file src/version.h.

At this point, you should return to completing the installation of **Bacula**.

50.2 Installing and Configuring SQLite – Phase II

This phase is done **after** you have run the **./configure** command to configure **Bacula**.

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_sqlite_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.

At this point, you can create the SQLite database and tables:

1. cd <install-directory>

   This directory contains the Bacula catalog interface routines.

2. ./make_sqlite_tables

   This script creates the SQLite database as well as the tables used by **Bacula**. This script will be automatically setup by the **./configure** program to create a database named **bacula.db** in **Bacula's** working directory.

50.3 Linking Bacula with SQLite

If you have followed the above steps, this will all happen automatically and the SQLite libraries will be linked into **Bacula**.

50.4 Testing SQLite

We have much less "production" experience using SQLite than using MySQL. SQLite has performed flawlessly for us in all our testing. However, several users have reported corrupted databases while using SQLite. For that reason, we do not recommend it for production use.

If Bacula crashes with the following type of error when it is started:

```
Using default Catalog name=MyCatalog DB=bacula
Could not open database "bacula".
sqlite.c:151 Unable to open Database=/var/lib/bacula/bacula.db.
ERR=malformed database schema - unable to open a temporary database file
for storing temporary tables
```

this is most likely caused by the fact that some versions of SQLite attempt to create a temporary file in the current directory. If that fails, because Bacula does not have write permission on the current directory, then you may get this errr. The solution is to start Bacula in a current directory where it has write permission.

50.5 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_sqlite_tables
./make_sqlite_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.

The internal database is not supported, please do not use it.

50.6 Internal Bacula Database

Previously it was intended to be used primarily by Bacula developers for testing; although SQLite is also a good choice for this. We do not recommend its use in general.

This database is simplistic in that it consists entirely of Bacula's internal structures appended sequentially to a file. Consequently, it is in most cases inappropriate for sites with many clients or systems with large numbers of files, or long-term production environments.

Below, you will find a table comparing the features available with SQLite and MySQL and with the internal Bacula database. At the current time, you cannot dynamically switch from one to the other, but must rebuild the Bacula source code. If you wish to experiment with both, it is possible to build both versions of Bacula and install them into separate directories.

Feature	**SQLite or MySQL**	**Bacula**
Job Record	Yes	Yes
Media Record	Yes	Yes
FileName Record	Yes	No
File Record	Yes	No
FileSet Record	Yes	Yes
Pool Record	Yes	Yes
Client Record	Yes	Yes
JobMedia Record	Yes	Yes
List Job Records	Yes	Yes
List Media Records	Yes	Yes
List Pool Records	Yes	Yes
List JobMedia Records	Yes	Yes
Delete Pool Record	Yes	Yes
Delete Media Record	Yes	Yes
Update Pool Record	Yes	Yes
Implement Verify	Yes	No
MD5 Signatures	Yes	No

In addition, since there is no SQL available, the Console commands: **sqlquery**, **query**, **retention**, and any other command that directly uses SQL are not available with the Internal database.

Chapter 51

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.bacula.org.

51.1 FDL

The GNU Free Documentation License (FDL) is used for this manual, which is a free and open license. This means that you may freely reproduce it and even make changes to it. However, rather than distribute your own version of this manual, we would much prefer if you would send any corrections or changes to the Bacula project.

The most recent version of the manual can always be found online at http://www.bacula.org.

51.2 GPL

The vast bulk of the source code is released under the GNU General Public License version 2..

Most of this code is copyrighted: Copyright ©2000-2007 Free Software Foundation Europe e.V.

Portions may be copyrighted by other people (ATT, the Free Software Foundation, ...). These files are released under the GPL license.

51.3 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.

51.4 Public Domain

Some of the Bacula code, or code that Bacula references, has been released to the public domain. E.g. md5.c, SQLite.

51.5 Trademark

Bacula® is a registered trademark of Kern Sibbald.

We have trademarked the Bacula name to ensure that any program using the name Bacula will be exactly compatible with the program that we have released. The use of the name Bacula is restricted to software systems that agree exactly with the program presented here.

51.6 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) 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 should be filled out then sent to:

Free Software Foundation Europe

Freedom Task Force
Sumatrastrasse 25
8006 Zürich
Switzerland

Please note that the above address is different from the officially registered office mentioned in the document. When you send in such a complete document, please notify me: kern at sibbald dot com.

51.7 Disclaimer

Chapter 52

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

GNU General Public License

image of a Philosophical GNU

- What to do if you see a possible GPL violation
- Translations of the GPL

52.1 Table of Contents

52.2 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

52.3 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

52.4 TERMS AND CONDITIONS

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. 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.

12. 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.

END OF TERMS AND CONDITIONS

52.5 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
{\em one line to give the program's name and an idea of what it does.}
Copyright (C) {\em yyyy}  {\em name of author}
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) {\em year} {\em name of author}
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `'show w'` and `'show c'` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `'show w'` and `'show c'`; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.
{\em signature of Ty Coon}, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License. Return to GNU's home page.

FSF & GNU inquiries & questions to gnu@gnu.org. Other ways to contact the FSF.

Comments on these web pages to webmasters@www.gnu.org, send other questions to gnu@gnu.org.

Updated: 3 Jan 2000 rms

GNU Lesser General Public License

image of a Philosophical GNU [English — Japanese]

- Why you shouldn't use the Lesser GPL for your next library
- What to do if you see a possible LGPL violation
- Translations of the LGPL
- The GNU Lesser General Public License as a text file
- The GNU Lesser General Public License as a Texinfo file

This GNU Lesser General Public License counts as the successor of the GNU Library General Public License. For an explanation of why this change was necessary, read the Why you shouldn't use the Lesser GPL for your next library article.

52.6 Table of Contents

52.7 GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

52.8 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

52.9 TERMS AND CONDITIONS

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** The modified work must itself be a software library.
- **b)** You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- **c)** You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- **d)** If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

  (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

  These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

  Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- **a)** Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

- **b)** Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

- **c)** Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- **d)** If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- **e)** Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- **a)** Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- **b)** Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "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 LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. 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 LIBRARY 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 LIBRARY (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 LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

52.10 How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
{\it one line to give the library's name and an idea of what it does.}
Copyright (C) {\it year}  {\it name of author}
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
```

```
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in
the library "Frob" (a library for tweaking knobs) written
by James Random Hacker.
{\it signature of Ty Coon}, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it! Return to GNU's home page.

FSF & GNU inquiries & questions to gnu@gnu.org. Other ways to contact the FSF.

Comments on these web pages to webmasters@www.gnu.org, send other questions to gnu@gnu.org.

Updated: 27 Nov 2000 paulv

Chapter 53

Bacula Projects

Once a new major version of Bacula is released, the Bacula users will vote on a list of new features. This vote is used as the main element determining what new features will be implemented for the next version. Generally, the development time for a new release is between four to nine months. Sometimes it may be a bit longer, but in that case, there will be a number of bug fix updates to the currently released version.

For the current list of project, please see the projects page in the CVS at: http://cvs.sourceforge.net/viewcvs.py/*checkout*/bacula/bacula/projects see the **projects** file in the main source directory. The projects file is updated approximately once every six months.

Separately from the project list, Kern maintains a current list of tasks as well as ideas, feature requests, and occasionally design notes. This list is updated roughly weekly (sometimes more often). For a current list of tasks you can see **kernstodo** in the Source Forge CVS at http://cvs.sourceforge.net/viewcvs.py/*checkout*/bacula/bacula/kernstodo.

Chapter 54

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 Win32 Bacula emulation code and for contributing it to the Bacula project.

Thanks to Robert Nelson for bringing our Win32 implementation up to par with all the same features that exist in the Unix/Linux versions. In addition, he has ported the Director and Storage daemon to Win32!

Thanks to Thorsten Engel for his excellent knowledge of Win32 systems, and for making the Win32 File daemon Unicode compatible, as well as making the Win32 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 (eay@cryptsoft.com) and also software written by Tim Hudson (tjh@cryptsoft.com).

The Bat (Bacula Administration Tool) graphs are based in part on the work of the Qwt project (http://qwt.sf.net).

The original variable expansion code used in the LabelFormat comes from the Open Source Software Project (www.ossp.org). 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

54.1 Bacula Bugs

Well fortunately there are not too many bugs, but thanks to Dan Langille, we have a bugs database 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.

Chapter 55

Variable Expansion

Please note that as of version 1.37, the Variable Expansion is deprecated and replaced by Python scripting (not yet documented).

Variable expansion is somewhat similar to Unix shell variable expansion. Currently (version 1.31), it is used only in format labels, but in the future, it will most likely be used in more places.

55.1 General Functionality

This is basically a string expansion capability that permits referencing variables, indexing arrays, conditional replacement of variables, case conversion, substring selection, regular expression matching and replacement, character class replacement, padding strings, repeated expansion in a user controlled loop, support of arithmetic expressions in the loop start, step and end conditions, and recursive expansion.

When using variable expansion characters in a Volume Label Format record, the format should always be enclosed in double quotes (").

For example, **${HOME}** will be replaced by your home directory as defined in the environment. If you have defined the variable **xxx** to be **Test**, then the reference **${xxx:p/7/Y/r}** will right pad the contents of **xxx** to a length of seven characters filling with the character **Y** giving **YYYTest**.

55.2 Bacula Variables

Within Bacula, there are three main classes of variables with some minor variations within the classes. The classes are:

Counters Counters are defined by the **Counter** resources in the Director's conf file. The counter can either be a temporary counter that lasts for the duration of Bacula's execution, or it can be a variable that is stored in the catalog, and thus retains its value from one Bacula execution to another. Counter variables may be incremented by postfixing a plus sign (+ after the variable name).

Internal Variables Internal variables are read-only, and may be related to the current job (i.e. Job name), or maybe special variables such as the date and time. The following variables are available:

Year – the full year

Month – the current month 1-12

Day – the day of the month 1-31

Hour – the hour 0-24

Minute – the current minute 0-59

Second – the current second 0-59

WeekDay – the current day of the week 0-6 with 0 being Sunday

Job – the job name

Dir – the Director's name

Level – the Job Level

Type – the Job type

JobId – the JobId

JobName – the unique job name composed of Job and date

Storage – the Storage daemon's name

Client – the Client's name

NumVols – the current number of Volumes in the Pool

Pool – the Pool name

Catalog – the Catalog name

MediaType – the Media Type

Environment Variables Environment variables are read-only, and must be defined in the environment prior to executing Bacula. Environment variables may be either scalar or an array, where the elements of the array are referenced by subscripting the variable name (e.g. **${Months[3]}**). Environment variable arrays are defined by separating the elements with a vertical bar (—), thus **set Months="Jan—Feb—Mar—Apr—..."** defines an environment variable named **Month** that will be treated as an array, and the reference **${Months[3]}** will yield **Mar**. The elements of the array can have differing lengths.

55.3 Full Syntax

Since the syntax is quite extensive, below, you will find the pseudo BNF. The special characters have the following meaning:

```
::=     definition
( )     grouping if the parens are not quoted
|       separates alternatives
'/'     literal / (or any other character)
CAPS    a character or character sequence
*       preceding item can be repeated zero or more times
?       preceding item can appear zero or one time
+       preceding item must appear one or more times
```

And the pseudo BNF describing the syntax is:

```
input        ::= ( TEXT
                 | variable
                 | INDEX_OPEN input INDEX_CLOSE (loop_limits)?
                 )*
variable     ::= DELIM_INIT (name|expression)
name         ::= (NAME_CHARS)+
expression   ::= DELIM_OPEN
                 (name|variable)+
                 (INDEX_OPEN num_exp INDEX_CLOSE)?
                 (':' command)*
                 DELIM_CLOSE
command      ::= '-' (TEXT_EXP|variable)+
               | '+' (TEXT_EXP|variable)+
               | 'o' NUMBER ('-'|',') (NUMBER)?
               | '#'
               | '*' (TEXT_EXP|variable)+
               | 's' '/' (TEXT_PATTERN)+
                     '/' (variable|TEXT_SUBST)*
```

```
                   '/' ('m'|'g'|'i'|'t')*
               | 'y' '/' (variable|TEXT_SUBST)+
                   '/' (variable|TEXT_SUBST)*
                   '/'
               | 'p' '/' NUMBER
                   '/' (variable|TEXT_SUBST)*
                   '/' ('r'|'l'|'c')
               | '%' (name|variable)+
                   ('(' (TEXT_ARGS)? ')')?
               | 'l'
               | 'u'
num_exp      ::= operand
               | operand ('+'|'-'|'*'|'/'|'%') num_exp
operand      ::= ('+'|'-')? NUMBER
               | INDEX_MARK
               | '(' num_exp ')'
               | variable
loop_limits ::= DELIM_OPEN
                (num_exp)? ',' (num_exp)? (',' (num_exp)?)?
                DELIM_CLOSE
NUMBER       ::= ('0'|...|'9')+
TEXT_PATTERN::= (^('/'))+
TEXT_SUBST   ::= (^(DELIM_INIT|'/'))+
TEXT_ARGS    ::= (^(DELIM_INIT|')'))+
TEXT_EXP     ::= (^(DELIM_INIT|DELIM_CLOSE|':'|'+'))+
TEXT         ::= (^(DELIM_INIT|INDEX_OPEN|INDEX_CLOSE))+
DELIM_INIT   ::= '$'
DELIM_OPEN   ::= '{'
DELIM_CLOSE ::= '}'
INDEX_OPEN   ::= '['
INDEX_CLOSE ::= ']'
INDEX_MARK   ::= '#'
NAME_CHARS   ::= 'a'|...|'z'|'A'|...|'Z'|'0'|...|'9'
```

55.4 Semantics

The items listed in **command** above, which always follow a colon (**:**) have the following meanings:

```
-    perform substitution if variable is empty
+    perform substitution if variable is not empty
o    cut out substring of the variable value
#    length of the variable value
*    substitute empty string if the variable value is not empty,
     otherwise substitute the trailing parameter
s    regular expression search and replace. The trailing
     options are: m = multiline, i = case insensitive,
                  g = global,    t = plain text (no regexp)
y    transpose characters from class A to class B
p    pad variable to l = left, r = right or c = center,
```

```
     with second value.
 %   special function call (none implemented)
 l   lower case the variable value
 u   upper case the variable value
```

The **loop_limits** are start, step, and end values.

A counter variable name followed immediately by a plus (+) will cause the counter to be incremented by one.

55.5 Examples

To create an ISO date:

```
DLT-${Year}-${Month:p/2/0/r}-${Day:p/2/0/r}
```

on 20 June 2003 would give **DLT-2003-06-20**

If you set the environment variable **mon** to

```
 January|February|March|April|May|...
 File-${mon[${Month}]}/${Day}/${Year}
```

on the first of March would give **File-March/1/2003**

Chapter 56

Using Stunnel to Encrypt Communications

Prior to version 1.37, Bacula did not have built-in communications encryption. Please see the TLS chapter if you are using Bacula 1.37 or greater.

Without too much effort, it is possible to encrypt the communications between any of the daemons. This chapter will show you how to use **stunnel** to encrypt communications to your client programs. We assume the Director and the Storage daemon are running on one machine that will be called **server** and the Client or File daemon is running on a different machine called **client**. Although the details may be slightly different, the same principles apply whether you are encrypting between Unix, Linux, or Win32 machines. This example was developed between two Linux machines running stunnel version 4.04-4 on a Red Hat Enterprise 3.0 system.

56.1 Communications Ports Used

First, you must know that with the standard Bacula configuration, the Director will contact the File daemon on port 9102. The File daemon then contacts the Storage daemon using the address and port parameters supplied by the Director. The standard port used will be 9103. This is the typical server/client view of the world, the File daemon is a server to the Director (i.e. listens for the Director to contact it), and the Storage daemon is a server to the File daemon.

56.2 Encryption

The encryption is accomplished between the Director and the File daemon by using an stunnel on the Director's machine (server) to encrypt the data and to contact an stunnel on the File daemon's machine (client), which decrypts the data and passes it to the client.

Between the File daemon and the Storage daemon, we use an stunnel on the File daemon's machine to encrypt the data and another stunnel on the Storage daemon's machine to decrypt the data.

As a consequence, there are actually four copies of stunnel running, two on the server and two on the client. This may sound a bit complicated, but it really isn't. To accomplish this, we will need to construct four separate conf files for stunnel, and we will need to make some minor modifications to the Director's conf file. None of the other conf files need to be changed.

56.3 A Picture

Since pictures usually help a lot, here is an overview of what we will be doing. Don't worry about all the details of the port numbers and such for the moment.

```
File daemon (client):
               stunnel-fd1.conf
                 |===========|
Port 29102  >----| Stunnel 1 |-----> Port 9102
                 |===========|
               stunnel-fd2.conf
                 |===========|
Port 9103   >----| Stunnel 2 |-----> server:29103
                 |===========|
Director (server):
               stunnel-dir.conf
                 |===========|
Port 29102  >----| Stunnel 3 |-----> client:29102
                 |===========|
               stunnel-sd.conf
                 |===========|
Port 29103  >----| Stunnel 4 |-----> 9103
                 |===========|
```

56.4 Certificates

In order for stunnel to function as a server, which it does in our diagram for Stunnel 1 and Stunnel 4, you must have a certificate and the key. It is possible to keep the two in separate files, but normally, you keep them in one single .pem file. You may create this certificate yourself in which case, it will be self-signed, or you may have it signed by a CA.

If you want your clients to verify that the server is in fact valid (Stunnel 2 and Stunnel 3), you will need to have the server certificates signed by a CA (Certificate Authority), and you will need to have the CA's public certificate (contains the CA's public key).

Having a CA signed certificate is **highly** recommended if you are using your client across the Internet, otherwise you are exposed to the man in the middle attack and hence loss of your data.

See below for how to create a self-signed certificate.

56.5 Securing the Data Channel

To simplify things a bit, let's for the moment consider only the data channel. That is the connection between the File daemon and the Storage daemon, which takes place on port 9103. In fact, in a minimalist solution, this is the only connection that needs to be encrypted, because it is the one that transports your data. The connection between the Director and the File daemon is simply a control channel used to start the job and get the job status.

Normally the File daemon will contact the Storage daemon on port 9103 (supplied by the Director), so we need an stunnel that listens on port 9103 on the File daemon's machine, encrypts the data and sends it to the Storage daemon. This is depicted by Stunnel 2 above. Note that this stunnel is listening on port 9103 and sending to server:29103. We use port 29103 on the server because if we would send the data to port 9103, it would go directly to the Storage daemon, which doesn't understand encrypted data. On the server machine, we run Stunnel 4, which listens on port 29103, decrypts the data and sends it to the Storage daemon, which is listening on port 9103.

56.6 Data Channel Configuration

The Storage resource of the bacula-dir.conf normally looks something like the following:

```
Storage {
  Name = File
  Address = server
  SDPort = 9103
  Password = storage_password
  Device = File
  Media Type = File
}
```

Notice that this is running on the server machine, and it points the File daemon back to server:9103, which is where our Storage daemon is listening. We modify this to be:

```
Storage {
  Name = File
  Address = localhost
  SDPort = 9103
  Password = storage_password
  Device = File
  Media Type = File
}
```

This causes the File daemon to send the data to the stunnel running on localhost (the client machine). We could have used client as the address as well.

56.7 Stunnel Configuration for the Data Channel

In the diagram above, we see above Stunnel 2 that we use stunnel-fd2.conf on the client. A pretty much minimal config file would look like the following:

```
client = yes
[29103]
accept = localhost:9103
connect = server:29103
```

The above config file does encrypt the data but it does not require a certificate, so it is subject to the man in the middle attack. The file I actually used, stunnel-fd2.conf, looked like this:

```
#
# Stunnel conf for Bacula client -> SD
#
pid = /home/kern/bacula/bin/working/stunnel.pid
#
# A cert is not mandatory here. If verify=2, a
#  cert signed by a CA must be specified, and
#  either CAfile or CApath must point to the CA's
#  cert
#
cert = /home/kern/stunnel/stunnel.pem
CAfile = /home/kern/ssl/cacert.pem
verify = 2
client = yes
# debug = 7
# foreground = yes
[29103]
accept = localhost:9103
connect = server:29103
```

You will notice that I specified a pid file location because I ran stunnel under my own userid so I could not use the default, which requires root permission. I also specified a certificate that I have as well as verify level 2 so that the certificate is required and verified, and I must supply the location of the CA (Certificate Authority) certificate so that the stunnel certificate can be verified. Finally, you will see that there are two lines commented out, which when enabled, produce a lot of nice debug info in the command window.

If you do not have a signed certificate (stunnel.pem), you need to delete the cert, CAfile, and verify lines.

Note that the stunnel.pem, is actually a private key and a certificate in a single file. These two can be kept and specified individually, but keeping them in one file is more convenient.

The config file, stunnel-sd.conf, needed for Stunnel 4 on the server machine is:

```
#
# Bacula stunnel conf for Storage daemon
#
pid = /home/kern/bacula/bin/working/stunnel.pid
#
# A cert is mandatory here, it may be self signed
#  If it is self signed, the client may not use
#  verify
#
cert   = /home/kern/stunnel/stunnel.pem
client = no
```

```
# debug = 7
# foreground = yes
[29103]
accept = 29103
connect = 9103
```

56.8 Starting and Testing the Data Encryption

It will most likely be the simplest to implement the Data Channel encryption in the following order:

- Setup and run Bacula backing up some data on your client machine without encryption.
- Stop Bacula.
- Modify the Storage resource in the Director's conf file.
- Start Bacula
- Start stunnel on the server with:

  ```
   stunnel stunnel-sd.conf
  ```

- Start stunnel on the client with:

  ```
  stunnel stunnel-fd2.conf
  ```

- Run a job.
- If it doesn't work, turn debug on in both stunnel conf files, restart the stunnels, rerun the job, repeat until it works.

56.9 Encrypting the Control Channel

The Job control channel is between the Director and the File daemon, and as mentioned above, it is not really necessary to encrypt, but it is good practice to encrypt it as well. The two stunnels that are used in this case will be Stunnel 1 and Stunnel 3 in the diagram above. Stunnel 3 on the server might normally listen on port 9102, but if you have a local File daemon, this will not work, so we make it listen on port 29102. It then sends the data to client:29102. Again we use port 29102 so that the stunnel on the client machine can decrypt the data before passing it on to port 9102 where the File daemon is listening.

56.10 Control Channel Configuration

We need to modify the standard Client resource, which would normally look something like:

```
Client {
  Name = client-fd
  Address = client
  FDPort = 9102
  Catalog = BackupDB
  Password = "xxx"
}
```

to be:

```
Client {
  Name = client-fd
  Address = localhost
  FDPort = 29102
  Catalog = BackupDB
  Password = "xxx"
}
```

This will cause the Director to send the control information to localhost:29102 instead of directly to the client.

56.11 Stunnel Configuration for the Control Channel

The stunnel config file, stunnel-dir.conf, for the Director's machine would look like the following:

```
#
# Bacula stunnel conf for the Directory to contact a client
#
```

```
pid = /home/kern/bacula/bin/working/stunnel.pid
#
# A cert is not mandatory here. If verify=2, a
#  cert signed by a CA must be specified, and
#  either CAfile or CApath must point to the CA's
#  cert
#
cert   = /home/kern/stunnel/stunnel.pem
CAfile = /home/kern/ssl/cacert.pem
verify = 2
client = yes
# debug = 7
# foreground = yes
[29102]
accept = localhost:29102
connect = client:29102
```

and the config file, stunnel-fd1.conf, needed to run stunnel on the Client would be:

```
#
# Bacula stunnel conf for the Directory to contact a client
#
pid = /home/kern/bacula/bin/working/stunnel.pid
#
# A cert is not mandatory here. If verify=2, a
#  cert signed by a CA must be specified, and
#  either CAfile or CApath must point to the CA's
#  cert
#
cert   = /home/kern/stunnel/stunnel.pem
CAfile = /home/kern/ssl/cacert.pem
verify = 2
client = yes
# debug = 7
# foreground = yes
[29102]
accept = localhost:29102
connect = client:29102
```

56.12 Starting and Testing the Control Channel

It will most likely be the simplest to implement the Control Channel encryption in the following order:

- Stop Bacula.
- Modify the Client resource in the Director's conf file.

- Start Bacula
- Start stunnel on the server with:

```
stunnel stunnel-dir.conf
```

- Start stunnel on the client with:

```
stunnel stunnel-fd1.conf
```

- Run a job.
- If it doesn't work, turn debug on in both stunnel conf files, restart the stunnels, rerun the job, repeat until it works.

56.13 Using stunnel to Encrypt to a Second Client

On the client machine, you can just duplicate the setup that you have on the first client file for file and it should work fine.

In the bacula-dir.conf file, you will want to create a second client pretty much identical to how you did for the first one, but the port number must be unique. We previously used:

```
Client {
  Name = client-fd
  Address = localhost
  FDPort = 29102
  Catalog = BackupDB
  Password = "xxx"
}
```

so for the second client, we will, of course, have a different name, and we will also need a different port. Remember that we used port 29103 for the Storage daemon, so for the second client, we can use port 29104, and the Client resource would look like:

```
Client {
  Name = client2-fd
  Address = localhost
  FDPort = 29104
  Catalog = BackupDB
  Password = "yyy"
}
```

Now, fortunately, we do not need a third stunnel to on the Director's machine, we can just add the new port to the config file, stunnel-dir.conf, to make:

```
#
# Bacula stunnel conf for the Directory to contact a client
#
pid = /home/kern/bacula/bin/working/stunnel.pid
#
# A cert is not mandatory here. If verify=2, a
#  cert signed by a CA must be specified, and
#  either CAfile or CApath must point to the CA's
#  cert
#
cert   = /home/kern/stunnel/stunnel.pem
CAfile = /home/kern/ssl/cacert.pem
verify = 2
client = yes
# debug = 7
# foreground = yes
[29102]
accept = localhost:29102
connect = client:29102
[29104]
accept = localhost:29102
connect = client2:29102
```

There are no changes necessary to the Storage daemon or the other stunnel so that this new client can talk to our Storage daemon.

56.14 Creating a Self-signed Certificate

You may create a self-signed certificate for use with stunnel that will permit you to make it function, but will not allow certificate validation. The .pem file containing both the certificate and the key can be made with the following, which I put in a file named **makepem**:

```
#!/bin/sh
#
# Simple shell script to make a .pem file that can be used
# with stunnel and Bacula
#
OPENSSL=openssl
   umask 77
   PEM1="/bin/mktemp openssl.XXXXXX"
   PEM2="/bin/mktemp openssl.XXXXXX"
```

```
${OPENSSL} req -newkey rsa:1024 -keyout $PEM1 -nodes \
    -x509 -days 365 -out $PEM2
cat $PEM1 > stunnel.pem
echo ""   >>stunnel.pem
cat $PEM2 >>stunnel.pem
rm $PEM1 $PEM2
```

The above script will ask you 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.

56.15 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 quite 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 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 explains how to do it, or you can read the documentation provided in the Open-source PKI Book project at Source Forge: http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm. Note, this link may change.

56.16 Using ssh to Secure the Communications

Please see the script **ssh-tunnel.sh** in the **examples** directory. It was contributed by Stephan Holl.

General Index

Director Index

File Daemon Index

Storage Daemon Index

Console Index