



Bacula® Concepts and Overview Guide

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

Kern Sibbald

February 19, 2010

This manual documents Bacula version 3.0.2 (18 July 2009)

Copyright © 1999-2009, Free Software Foundation Europe e.V.

Bacula ® is a registered trademark of Kern Sibbald.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 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".

Contents

1	What is Bacula?	1
1.1	Who Needs Bacula?	1
1.2	Bacula Components or Services	1
1.3	Bacula Configuration	4
1.4	Conventions Used in this Document	4
1.5	Quick Start	5
1.6	Terminology	5
1.7	What Bacula is Not	7
1.8	Interactions Between the Bacula Services	8
2	New Features in 3.0.2	9
2.1	Full restore from a given JobId	9
2.2	Source Address	10
2.3	Show volume availability when doing restore	10
2.4	Accurate estimate command	10
3	New Features in 3.0.0	13
3.1	Accurate Backup	13
3.1.1	Accurate = <yes—no>	13
3.2	Copy Jobs	13
3.3	ACL Updates	16
3.4	Extended Attributes	17
3.5	Shared objects	18
3.6	Building Static versions of Bacula	18
3.7	Virtual Backup (Vbackup)	18
3.8	Catalog Format	20
3.9	64 bit Windows Client	20

3.10 Duplicate Job Control	21
3.10.1 Allow Duplicate Jobs = <yes—no>	21
3.10.2 Allow Higher Duplicates = <yes—no>	21
3.10.3 Cancel Queued Duplicates = <yes—no>	22
3.10.4 Cancel Running Duplicates = <yes—no>	22
3.11 TLS Authentication	22
3.11.1 TLS Authenticate = yes	22
3.12 bextract non-portable Win32 data	22
3.13 State File updated at Job Termination	22
3.14 MaxFullInterval = <time-interval>	23
3.15 MaxDiffInterval = <time-interval>	23
3.16 Honor No Dump Flag = <yes—no>	23
3.17 Exclude Dir Containing = <filename-string>	23
3.18 Bacula Plugins	24
3.18.1 Plugin Directory	24
3.18.2 Plugin Options	24
3.18.3 Plugin Options ACL	24
3.18.4 Plugin = <plugin-command-string>	24
3.19 The bpipe Plugin	25
3.20 Microsoft Exchange Server 2003/2007 Plugin	26
3.20.1 Background	26
3.20.2 Concepts	26
3.20.3 Installing	26
3.20.4 Backup up	26
3.20.5 Restoring	27
3.20.6 Restoring to the Recovery Storage Group	28
3.20.7 Restoring on Microsoft Server 2007	28
3.20.8 Caveats	28
3.21 libdbi Framework	28
3.22 Console Command Additions and Enhancements	30
3.22.1 Display Autochanger Content	30
3.22.2 list joblog job=xxx or jobid=nnn	30
3.22.3 Use separator for multiple commands	30

3.22.4	Deleting Volumes	30
3.23	Bare Metal Recovery	31
3.24	Miscellaneous	31
3.24.1	Allow Mixed Priority = <yes—no>	31
3.24.2	Bootstrap File Directive – FileRegex	32
3.24.3	Bootstrap File Optimization Changes	32
3.24.4	Solaris ZFS/NFSv4 ACLs	32
3.24.5	Virtual Tape Emulation	32
3.24.6	Bat Enhancements	32
3.24.7	RunScript Enhancements	33
3.24.8	Status Enhancements	33
3.24.9	Connect Timeout	33
3.24.10	truncate for NFS Volumes	33
3.24.11	Support for Ubuntu	33
3.24.12	Recycle Pool = <pool-name>	33
3.24.13	FD Version	34
3.24.14	Max Run Sched Time = <time-period-in-seconds>	34
3.24.15	Max Wait Time = <time-period-in-seconds>	34
3.24.16	Incremental—Differential Max Wait Time = <time-period-in-seconds>	34
3.24.17	Max Run Time directives	34
3.24.18	Statistics Enhancements	35
3.24.19	ScratchPool = <pool-resource-name>	36
3.24.20	Enhanced Attribute Despooling	36
3.24.21	SpoolSize = <size-specification-in-bytes>	36
3.24.22	MaxConsoleConnections = <number>	36
3.24.23	VerId = <string>	37
3.24.24	dbcheck enhancements	37
3.24.25	--docdir configure option	37
3.24.26	--htmldir configure option	37
3.24.27	--with-pluginindir configure option	37
4	Bacula FD Plugin API	39
4.1	Normal vs Command Plugins	39

4.2	Loading Plugins	40
4.3	loadPlugin	40
4.4	Plugin Entry Points	43
4.4.1	newPlugin(bpContext *ctx)	43
4.4.2	freePlugin(bpContext *ctx)	43
4.4.3	getPluginValue(bpContext *ctx, pVariable var, void *value)	43
4.4.4	setPluginValue(bpContext *ctx, pVariable var, void *value)	43
4.4.5	handlePluginEvent(bpContext *ctx, bEvent *event, void *value)	43
4.4.6	startBackupFile(bpContext *ctx, struct save_pkt *sp)	45
4.4.7	endBackupFile(bpContext *ctx)	46
4.4.8	startRestoreFile(bpContext *ctx, const char *cmd)	46
4.4.9	createFile(bpContext *ctx, struct restore_pkt *rp)	46
4.4.10	setFileAttributes(bpContext *ctx, struct restore_pkt *rp)	48
4.4.11	endRestoreFile(bpContext *ctx)	48
4.4.12	pluginIO(bpContext *ctx, struct io_pkt *io)	48
4.4.13	bool checkFile(bpContext *ctx, char *fname)	49
4.5	Bacula Plugin Entrypoints	49
4.5.1	bRC registerBaculaEvents(bpContext *ctx, ...)	49
4.5.2	bRC getBaculaValue(bpContext *ctx, bVariable var, void *value)	50
4.5.3	bRC setBaculaValue(bpContext *ctx, bVariable var, void *value)	50
4.5.4	bRC JobMessage(bpContext *ctx, const char *file, int line, int type, utime_t mtime, const char *fmt, ...) 50	
4.5.5	bRC DebugMessage(bpContext *ctx, const char *file, int line, int level, const char *fmt, ...) 50	
4.5.6	void baculaMalloc(bpContext *ctx, const char *file, int line, size_t size)	50
4.5.7	void baculaFree(bpContext *ctx, const char *file, int line, void *mem)	50
4.6	Building Bacula Plugins	50
5	The Current State of Bacula	53
5.1	What is Implemented	53
5.2	Advantages Over Other Backup Programs	55
5.3	Current Implementation Restrictions	55
5.4	Design Limitations or Restrictions	56
6	System Requirements	57
7	Supported Operating Systems	59

8	Supported Tape Drives	61
8.1	Unsupported Tape Drives	62
8.2	FreeBSD Users Be Aware!!!	62
8.3	Supported Autochangers	62
8.4	Tape Specifications	62
9	A Brief Tutorial	65
9.1	Before Running Bacula	65
9.2	Starting the Database	66
9.3	Starting the Daemons	66
9.4	Using the Director to Query and Start Jobs	66
9.5	Running a Job	68
9.6	Restoring Your Files	72
9.7	Quitting the Console Program	74
9.8	Adding a Second Client	74
9.9	When The Tape Fills	75
9.10	Other Useful Console Commands	77
9.11	Debug Daemon Output	77
9.12	Patience When Starting Daemons or Mounting Blank Tapes	78
9.13	Difficulties Connecting from the FD to the SD	78
9.14	Daemon Command Line Options	78
9.15	Creating a Pool	79
9.16	Labeling Your Volumes	79
9.17	Labeling Volumes with the Console Program	80
10	The Restore Command	83
10.1	General	83
10.2	The Restore Command	83
10.2.1	Restore a pruned job using a pattern	88
10.3	Selecting Files by Filename	88
10.4	Replace Options	89
10.5	Command Line Arguments	90
10.6	Using File Relocation	91
10.6.1	Introduction	91

10.6.2	RegexWhere Format	91
10.7	Restoring Directory Attributes	92
10.8	Restoring on Windows	92
10.9	Restoring Files Can Be Slow	93
10.10	Problems Restoring Files	93
10.11	Restore Errors	94
10.12	Example Restore Job Resource	94
10.13	File Selection Commands	94
10.14	Restoring When Things Go Wrong	96
11	Automatic Volume Recycling	101
11.1	Automatic Pruning	102
11.2	Pruning Directives	102
11.3	Recycling Algorithm	104
11.4	Recycle Status	105
11.5	Making Bacula Use a Single Tape	106
11.6	Daily, Weekly, Monthly Tape Usage Example	106
11.7	Automatic Pruning and Recycling Example	108
11.8	Manually Recycling Volumes	109
12	Basic Volume Management	111
12.1	Key Concepts and Resource Records	111
12.1.1	Pool Options to Limit the Volume Usage	112
12.1.2	Automatic Volume Labeling	113
12.1.3	Restricting the Number of Volumes and Recycling	113
12.2	Concurrent Disk Jobs	114
12.3	An Example	115
12.4	Backing up to Multiple Disks	117
12.5	Considerations for Multiple Clients	118
13	Automated Disk Backup	123
13.1	The Problem	123
13.2	The Solution	123
13.3	Overall Design	124
13.3.1	Full Pool	124

13.3.2 Differential Pool	125
13.3.3 Incremental Pool	125
13.4 The Actual Conf Files	125
14 Migration and Copy	129
14.1 Migration and Copy Job Resource Directives	130
14.2 Migration Pool Resource Directives	131
14.3 Important Migration Considerations	132
14.4 Example Migration Jobs	133
15 Backup Strategies	135
15.1 Simple One Tape Backup	135
15.1.1 Advantages	135
15.1.2 Disadvantages	135
15.1.3 Practical Details	135
15.2 Manually Changing Tapes	136
15.3 Daily Tape Rotation	136
15.3.1 Advantages	136
15.3.2 Disadvantages	137
15.3.3 Practical Details	137
16 Autochanger Support	141
16.1 Knowing What SCSI Devices You Have	142
16.2 Example Scripts	143
16.3 Slots	143
16.4 Multiple Devices	143
16.5 Device Configuration Records	144
17 Autochanger Resource	147
17.1 An Example Configuration File	148
17.2 A Multi-drive Example Configuration File	148
17.3 Specifying Slots When Labeling	149
17.4 Changing Cartridges	150
17.5 Dealing with Multiple Magazines	150
17.6 Simulating Barcodes in your Autochanger	151

17.7 The Full Form of the Update Slots Command	151
17.8 FreeBSD Issues	152
17.9 Testing Autochanger and Adapting mtx-changer script	152
17.10 Using the Autochanger	153
17.11 Barcode Support	154
17.12 Use bconsole to display Autochanger content	155
17.13 Bacula Autochanger Interface	155
18 Supported Autochangers	157
19 Data Spooling	161
19.1 Data Spooling Directives	161
19.2 !!! MAJOR WARNING !!!	162
19.3 Other Points	162
20 Using Bacula catalog to grab information	163
20.1 Job statistics	163
21 Python Scripting	165
21.1 Python Configuration	165
21.2 Bacula Events	166
21.3 Python Objects	166
21.4 Python Console Command	169
21.5 Debugging Python Scripts	169
21.6 Python Example	170
22 ANSI and IBM Tape Labels	173
22.1 Director Pool Directive	173
22.2 Storage Daemon Device Directives	173
23 The Windows Version of Bacula	175
23.1 Win32 Installation	175
23.2 Post Win32 Installation	179
23.3 Uninstalling Bacula on Win32	179
23.4 Dealing with Win32 Problems	179
23.5 Windows Compatibility Considerations	181

23.6	Volume Shadow Copy Service	182
23.7	VSS Problems	183
23.8	Windows Firewalls	184
23.9	Windows Port Usage	184
23.10	Windows Disaster Recovery	184
23.11	Windows Restore Problems	184
23.12	Windows Ownership and Permissions Problems	185
23.13	Manually resetting the Permissions	185
23.14	Backing Up the WinNT/XP/2K System State	187
23.15	Considerations for Filename Specifications	188
23.16	Win32 Specific File daemon Command Line	188
23.17	Shutting down Windows Systems	189
24	Disaster Recovery Using Bacula	191
24.1	General	191
24.2	Important Considerations	191
24.3	Steps to Take Before Disaster Strikes	191
24.4	Bare Metal Recovery on Linux with a Rescue CD	192
24.5	Requirements	192
24.6	Restoring a Client System	192
24.7	Boot with your Rescue CDROM	193
24.8	Restoring a Server	195
24.9	Linux Problems or Bugs	196
24.10	Bare Metal Recovery using a LiveCD	196
24.11	FreeBSD Bare Metal Recovery	197
24.12	Solaris Bare Metal Recovery	198
24.13	Preparing Solaris Before a Disaster	198
24.14	Bugs and Other Considerations	199
24.15	Disaster Recovery of Win32 Systems	199
24.16	Ownership and Permissions on Win32 Systems	199
24.17	Alternate Disaster Recovery Suggestion for Win32 Systems	200
24.18	Restoring to a Running System	200
24.19	Additional Resources	201

25 Bacula TLS – Communications Encryption	203
25.1 TLS Configuration Directives	203
25.2 Creating a Self-signed Certificate	204
25.3 Getting a CA Signed Certificate	205
25.4 Example TLS Configuration Files	205
26 Data Encryption	209
26.1 Building Bacula with Encryption Support	210
26.2 Encryption Technical Details	210
26.3 Decrypting with a Master Key	210
26.4 Generating Private/Public Encryption Keys	211
26.5 Example Data Encryption Configuration	211
27 Using Bacula to Improve Computer Security	213
27.1 The Details	213
27.2 Running the Verify	214
27.3 What To Do When Differences Are Found	215
27.4 A Verify Configuration Example	216
28 The Bootstrap File	219
28.1 Bootstrap File Format	219
28.2 Automatic Generation of Bootstrap Files	222
28.3 Bootstrap for bscan	223
28.4 A Final Bootstrap Example	223
29 Bacula Copyright, Trademark, and Licenses	225
29.1 FDL	225
29.2 GPL	225
29.3 LGPL	225
29.4 Public Domain	225
29.5 Trademark	226
29.6 Fiduciary License Agreement	226
29.7 Disclaimer	226
30 GNU Free Documentation License	227
30.1 Table of Contents	233

30.2 GNU GENERAL PUBLIC LICENSE	233
30.3 Preamble	233
30.4 TERMS AND CONDITIONS	234
30.5 How to Apply These Terms to Your New Programs	237
30.6 Table of Contents	238
30.7 GNU LESSER GENERAL PUBLIC LICENSE	238
30.8 Preamble	238
30.9 TERMS AND CONDITIONS	239
30.10How to Apply These Terms to Your New Libraries	244
31 Bacula Projects	245
32 Thanks	247
32.1 Bacula Bugs	249
33 Variable Expansion	251
33.1 General Functionality	251
33.2 Bacula Variables	251
33.3 Full Syntax	252
33.4 Semantics	253
33.5 Examples	253
34 Using Stunnel to Encrypt Communications	255
34.1 Communications Ports Used	255
34.2 Encryption	255
34.3 A Picture	256
34.4 Certificates	256
34.5 Securing the Data Channel	256
34.6 Data Channel Configuration	257
34.7 Stunnel Configuration for the Data Channel	257
34.8 Starting and Testing the Data Encryption	258
34.9 Encrypting the Control Channel	258
34.10Control Channel Configuration	259
34.11Stunnel Configuration for the Control Channel	259
34.12Starting and Testing the Control Channel	260

34.13	Using stunnel to Encrypt to a Second Client	260
34.14	Creating a Self-signed Certificate	261
34.15	Getting a CA Signed Certificate	262
34.16	Using ssh to Secure the Communications	262
35	DVD Volumes	263
35.1	DVD Specific SD Directives	263
35.2	Edit Codes for DVD Directives	264
35.3	DVD Specific Director Directives	265
35.4	Other Points	265

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 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.



niatis for this graphic and the one below)

(thanks to Aristedes Ma-

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

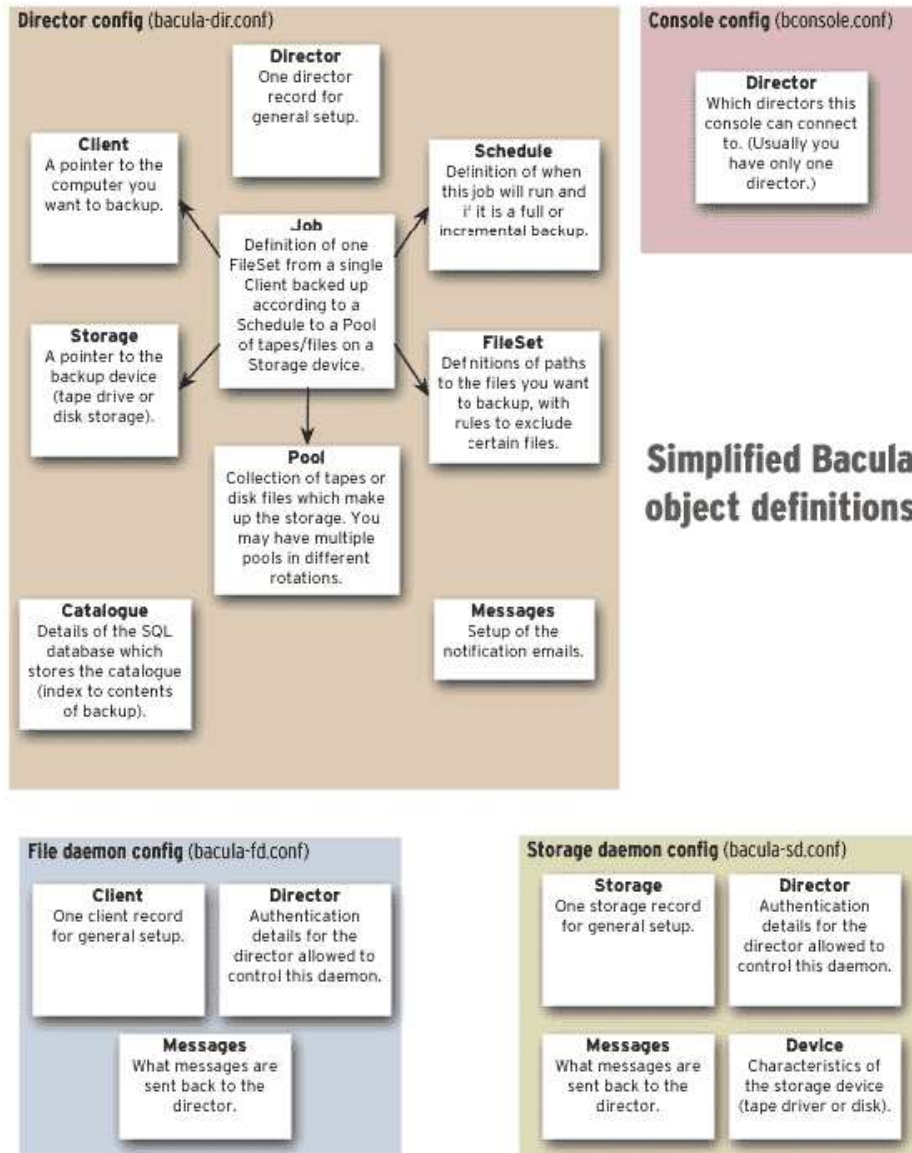
A Bacula Monitor service is the program that allows the administrator or user to watch current status of Bacula Directors, Bacula File Daemons and Bacula Storage Daemons. Currently, only a GTK+ version is available, which works with GNOME, KDE, or any window manager that supports the FreeDesktop.org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

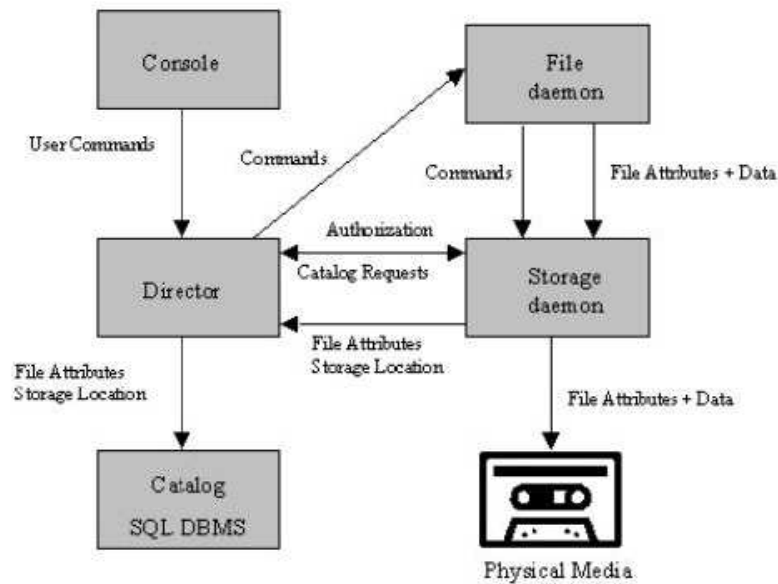
Bacula is a backup, restore and verification program and is not a complete disaster recovery system in itself, but it can be a key part of one if you plan carefully and follow the instructions included in the Disaster Recovery Chapter of this manual.

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, and/or a Bacula Rescue disk to save the current partitioning information of your hard disk, and maintain a complete Bacula backup, it is possible to completely recover your system from "bare metal" that is starting from an empty disk.

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

1.8 Interactions Between the Bacula Services

The 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.



Chapter 2

New Features in 3.0.2

This chapter presents the new features added to the development 3.0.2 versions to be released as Bacula version 3.0.2 sometime in 2009.

2.1 Full restore from a given JobId

This feature allows selecting a JobId and having Bacula automatically select all other jobs that comprise a full backup up to and including the selected JobId.

Assume we start with the following jobs:

jobid	client	starttime	level	jobfiles	jobbytes
6	localhost-fd	2009-07-15 11:45:49	I	2	0
5	localhost-fd	2009-07-15 11:45:45	I	15	44143
3	localhost-fd	2009-07-15 11:45:38	I	1	10
1	localhost-fd	2009-07-15 11:45:30	F	1527	44143073

Below is an example of selecting this new feature (which is number 12 in the menu).

```
* restore
To select the JobIds, you have the following choices:
  1: List last 20 Jobs run
  2: List Jobs where a given File is saved
...
 12: Select full restore to a specified JobId
 13: Cancel

Select item: (1-13): 12
Enter JobId to restore: 5
You have selected the following JobIds: 1,3,5

Building directory tree for JobId(s) 1,3,5 ... ++++++
1,444 files inserted into the tree.
```

This project was funded by Bacula Systems.

2.2 Source Address

A feature has been added which allows the administrator to specify the address from which the director and file daemons will attempt connections from. This may be used to simplify system configuration overhead when working in complex networks utilizing multi-homing and policy-routing.

To accomplish this, two new configuration directives have been implemented:

```
FileDaemon {
    FDSourceAddress=10.0.1.20    # Always initiate connections from this address
}

Director {
    DirSourceAddress=10.0.1.10   # Always initiate connections from this address
}
```

Simply adding specific host routes would have an undesirable side-effect: any application trying to contact the destination host would be forced to use the more specific route, possibly diverting management traffic onto a backup VLAN. Instead of adding host routes for each client connected to a multi-homed backup server (for example where there are management and backup VLANs), one can use the new directives to specify a specific source address at the application level.

Additionally, this allows the simplification and abstraction of firewall rules when dealing with a Hot-Standby director or storage daemon configuration. The Hot-standby pair may share a CARP address, which connections must be sourced from, while system services listen and act from the unique interface addresses.

This project was funded by Collaborative Fusion, Inc.

2.3 Show volume availability when doing restore

When doing a restore the restore selection dialog ends by displaying this screen:

```
The job will require the following
Volume(s)           Storage(s)           SD Device(s)
=====
*000741L3           LT0-4           LT03
*000866L3           LT0-4           LT03
*000765L3           LT0-4           LT03
*000764L3           LT0-4           LT03
*000756L3           LT0-4           LT03
*001759L3           LT0-4           LT03
*001763L3           LT0-4           LT03
 001762L3           LT0-4           LT03
 001767L3           LT0-4           LT03
```

Volumes marked with ‘*’ are online.

This should help getting large restores through minimizing the time spent waiting for operator to drop by and change tapes in the library.

This project was funded by Bacula Systems.

2.4 Accurate estimate command

The `estimate` command can now use the accurate code to detect changes and give a better estimation.

You can set the accurate behavior on command line using `accurate=yes/no` or use the Job setting as default value.

```
* estimate listing accurate=yes level=incremental job=BackupJob
```

This project was funded by Bacula Systems.

Chapter 3

New Features in 3.0.0

This chapter presents the new features added to the development 2.5.x versions to be released as Bacula version 3.0.0 sometime in April 2009.

3.1 Accurate Backup

As with most other backup programs, by default Bacula decides what files to backup for Incremental and Differential backup by comparing the change (`st_ctime`) and modification (`st_mtime`) times of the file to the time the last backup completed. If one of those two times is later than the last backup time, then the file will be backed up. This does not, however, permit tracking what files have been deleted and will miss any file with an old time that may have been restored to or moved onto the client filesystem.

3.1.1 Accurate = <yes—no>

If the **Accurate** = <yes—no> directive is enabled (default no) in the Job resource, the job will be run as an Accurate Job. For a **Full** backup, there is no difference, but for **Differential** and **Incremental** backups, the Director will send a list of all previous files backed up, and the File daemon will use that list to determine if any new files have been added or moved and if any files have been deleted. This allows Bacula to make an accurate backup of your system to that point in time so that if you do a restore, it will restore your system exactly.

One note of caution about using Accurate backup is that it requires more resources (CPU and memory) on both the Director and the Client machines to create the list of previous files backed up, to send that list to the File daemon, for the File daemon to keep the list (possibly very big) in memory, and for the File daemon to do comparisons between every file in the FileSet and the list. In particular, if your client has lots of files (more than a few million), you will need lots of memory on the client machine.

Accurate must not be enabled when backing up with a plugin that is not specially designed to work with Accurate. If you enable it, your restores will probably not work correctly.

This project was funded by Bacula Systems.

3.2 Copy Jobs

A new **Copy** job type 'C' has been implemented. It is similar to the existing Migration feature with the exception that the Job that is copied is left unchanged. This essentially creates two identical copies of the same backup. However, the copy is treated as a copy rather than a backup job, and hence is not directly available for restore. The **restore** command lists copy jobs and allows selection of copies by using `jobid=`

option. If the keyword **copies** is present on the command line, Bacula will display the list of all copies for selected jobs.

```
* restore copies
[...]
```

These JobIds have copies as follows:

```
+-----+-----+-----+-----+-----+-----+
| JobId | Job                               | CopyJobId | MediaType      |
+-----+-----+-----+-----+-----+-----+
| 2      | CopyJobSave.2009-02-17_16.31.00.11 | 7          | DiskChangerMedia |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| JobId | Level | JobFiles | JobBytes | StartTime           | VolumeName      |
+-----+-----+-----+-----+-----+-----+
| 19    | F     | 6274     | 76565018 | 2009-02-17 16:30:45 | ChangerVolume002 |
| 2     | I     | 1        | 5        | 2009-02-17 16:30:51 | FileVolume001    |
+-----+-----+-----+-----+-----+-----+
```

You have selected the following JobIds: 19,2

```
Building directory tree for JobId(s) 19,2 ... +-----+
5,611 files inserted into the tree.
...
```

The Copy Job runs without using the File daemon by copying the data from the old backup Volume to a different Volume in a different Pool. See the Migration documentation for additional details. For copy Jobs there is a new selection criterium named PoolUncopiedJobs which copies all jobs from a pool to an other pool which were not copied before. Next to that the client, volume, job or sql query are possible ways of selecting jobs which should be copied. Selection types like smallestvolume, oldestvolume, pooloccupancy and pooltime are probably more suited for migration jobs only. But we could imagine some people have a valid use for those kind of copy jobs too.

If bacula finds a copy when a job record is purged (deleted) from the catalog, it will promote the copy as *real* backup and will make it available for automatic restore. If more than one copy is available, it will promote the copy with the smallest jobid.

A nice solution which can be build with the new copy jobs is what is called the disk-to-disk-to-tape backup (DTDTT). A sample config could look somethings like the one below:

```
Pool {
    Name = FullBackupsVirtualPool
    Pool Type = Backup
    Purge Oldest Volume = Yes
    Storage = vtl
    NextPool = FullBackupsTapePool
}
```

```
Pool {
    Name = FullBackupsTapePool
    Pool Type = Backup
    Recycle = Yes
    AutoPrune = Yes
    Volume Retention = 365 days
    Storage = superloader
}
```

```
#
# Fake fileset for copy jobs
#
Fileset {
```

```

Name = None
Include {
    Options {
        signature = MD5
    }
}
}

#
# Fake client for copy jobs
#
Client {
    Name = None
    Address = localhost
    Password = "NoNe"
    Catalog = MyCatalog
}

#
# Default template for a CopyDiskToTape Job
#
JobDefs {
    Name = CopyDiskToTape
    Type = Copy
    Messages = StandardCopy
    Client = None
    FileSet = None
    Selection Type = PoolUncopiedJobs
    Maximum Concurrent Jobs = 10
    SpoolData = No
    Allow Duplicate Jobs = Yes
    Allow Higher Duplicates = No
    Cancel Queued Duplicates = No
    Cancel Running Duplicates = No
    Priority = 13
}

Schedule {
    Name = DaySchedule7:00
    Run = Level=Full daily at 7:00
}

Job {
    Name = CopyDiskToTapeFullBackups
    Enabled = Yes
    Schedule = DaySchedule7:00
    Pool = FullBackupsVirtualPool
    JobDefs = CopyDiskToTape
}

```

The example above had 2 pool which are copied using the PoolUncopiedJobs selection criteria. Normal Full backups go to the Virtual pool and are copied to the Tape pool the next morning.

The command `list copies [jobid=x,y,z]` lists copies for a given **jobid**.

```

*list copies
+-----+-----+-----+-----+
| JobId | Job                               | CopyJobId | MediaType |
+-----+-----+-----+-----+

```

3.3 ACL Updates

The whole ACL code had been overhauled and in this version each platforms has different streams for each type of acl available on such an platform. As ACLs between platforms tend to be not that portable (most implement POSIX acls but some use an other draft or a completely different format) we currently only allow certain platform specific ACL streams to be decoded and restored on the same platform that they were created on. The old code allowed to restore ACL cross platform but the comments already mention that not being to wise. For backward compatability the new code will accept the two old ACL streams and handle those with the platform specific handler. But for all new backups it will save the ACLs using the new streams.

Currently the following platforms support ACLs:

- **AIX**
- **Darwin/OSX**
- **FreeBSD**
- **HPUX**
- **IRIX**
- **Linux**
- **Tru64**
- **Solaris**

Currently we support the following ACL types (these ACL streams use a reserved part of the stream numbers):

- **STREAM_ACL_AIX_TEXT** 1000 AIX specific string representation from `acl.get`
- **STREAM_ACL_DARWIN_ACCESS_ACL** 1001 Darwin (OSX) specific `acl_t` string representation from `acl_to_text` (POSIX acl)
- **STREAM_ACL_FREEBSD_DEFAULT_ACL** 1002 FreeBSD specific `acl_t` string representation from `acl_to_text` (POSIX acl) for default acls.
- **STREAM_ACL_FREEBSD_ACCESS_ACL** 1003 FreeBSD specific `acl_t` string representation from `acl_to_text` (POSIX acl) for access acls.
- **STREAM_ACL_HPUX_ACL_ENTRY** 1004 HPUX specific `acl_entry` string representation from `acltostr` (POSIX acl)
- **STREAM_ACL_IRIX_DEFAULT_ACL** 1005 IRIX specific `acl_t` string representation from `acl_to_text` (POSIX acl) for default acls.
- **STREAM_ACL_IRIX_ACCESS_ACL** 1006 IRIX specific `acl_t` string representation from `acl_to_text` (POSIX acl) for access acls.
- **STREAM_ACL_LINUX_DEFAULT_ACL** 1007 Linux specific `acl_t` string representation from `acl_to_text` (POSIX acl) for default acls.
- **STREAM_ACL_LINUX_ACCESS_ACL** 1008 Linux specific `acl_t` string representation from `acl_to_text` (POSIX acl) for access acls.

- **STREAM_ACL_TRU64_DEFAULT_ACL** 1009 Tru64 specific `acl_t` string representation from `acl_to_text` (POSIX `acl`) for default acls.
- **STREAM_ACL_TRU64_DEFAULT_DIR_ACL** 1010 Tru64 specific `acl_t` string representation from `acl_to_text` (POSIX `acl`) for default acls.
- **STREAM_ACL_TRU64_ACCESS_ACL** 1011 Tru64 specific `acl_t` string representation from `acl_to_text` (POSIX `acl`) for access acls.
- **STREAM_ACL_SOLARIS_ACLNT** 1012 Solaris specific `aclnt_t` string representation from `acltotext` or `acl_totext` (POSIX `acl`)
- **STREAM_ACL_SOLARIS_ACE** 1013 Solaris specific `ace_t` string representation from `acl_totext` (NFSv4 or ZFS `acl`)

In future versions we might support conversion functions from one type of `acl` into an other for types that are either the same or easily convertible. For now the streams are separate and restoring them on a platform that doesn't recognize them will give you a warning.

3.4 Extended Attributes

Something that was on the project list for some time is now implemented for platforms that support a similar kind of interface. It's the support for backup and restore of so called extended attributes. As extended attributes are so platform specific these attributes are saved in separate streams for each platform. Restores of the extended attributes can only be performed on the same platform the backup was done. There is support for all types of extended attributes, but restoring from one type of filesystem onto an other type of filesystem on the same platform may lead to surprises. As extended attributes can contain any type of data they are stored as a series of so called value-pairs. This data must be seen as mostly binary and is stored as such. As security labels from `selinux` are also extended attributes this option also stores those labels and no specific code is enabled for handling `selinux` security labels.

Currently the following platforms support extended attributes:

- Darwin/OSX
- FreeBSD
- Linux
- NetBSD

On linux acls are also extended attributes, as such when you enable ACLs on a Linux platform it will NOT save the same data twice e.g. it will save the ACLs and not the same extended attribute.

To enable the backup of extended attributes please add the following to your fileset definition.

```
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
      xattrsupport = yes
    }
    File = ...
  }
}
```

3.5 Shared objects

A default build of Bacula will now create the libraries as shared objects (.so) rather than static libraries as was previously the case. The shared libraries are built using **libtool** so it should be quite portable.

An important advantage of using shared objects is that on a machine with the Directory, File daemon, the Storage daemon, and a console, you will have only one copy of the code in memory rather than four copies. Also the total size of the binary release is smaller since the library code appears only once rather than once for every program that uses it; this results in significant reduction in the size of the binaries particularly for the utility tools.

In order for the system loader to find the shared objects when loading the Bacula binaries, the Bacula shared objects must either be in a shared object directory known to the loader (typically /usr/lib) or they must be in the directory that may be specified on the `./configure` line using the `--libdir` option as:

```
./configure --libdir=/full-path/dir
```

the default is /usr/lib. If `--libdir` is specified, there should be no need to modify your loader configuration provided that the shared objects are installed in that directory (Bacula does this with the `make install` command). The shared objects that Bacula references are:

```
libbaccfg.so  
libbacfind.so  
libbacpy.so  
libbac.so
```

These files are symbolically linked to the real shared object file, which has a version number to permit running multiple versions of the libraries if desired (not normally the case).

If you have problems with libtool or you wish to use the old way of building static libraries, or you want to build a static version of Bacula you may disable libtool on the configure command line with:

```
./configure --disable-libtool
```

3.6 Building Static versions of Bacula

In order to build static versions of Bacula, in addition to configuration options that were needed you now must also add `--disable-libtool`. Example

```
./configure --enable-static-client-only --disable-libtool
```

3.7 Virtual Backup (Vbackup)

Bacula's virtual backup feature is often called Synthetic Backup or Consolidation in other backup products. It permits you to consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

In some respects the Vbackup feature works similar to a Migration job, in that Bacula normally reads the data from the pool specified in the Job resource, and writes it to the **Next Pool** specified in the Job

resource. Note, this means that usually the output from the Virtual Backup is written into a different pool from where your prior backups are saved. Doing it this way guarantees that you will not get a deadlock situation attempting to read and write to the same volume in the Storage daemon. If you then want to do subsequent backups, you may need to move the Virtual Full Volume back to your normal backup pool. Alternatively, you can set your **Next Pool** to point to the current pool. This will cause Bacula to read and write to Volumes in the current pool. In general, this will work, because Bacula will not allow reading and writing on the same Volume. In any case, once a VirtualFull has been created, and a restore is done involving the most current Full, it will read the Volume or Volumes by the VirtualFull regardless of in which Pool the Volume is found.

The Vbackup is enabled on a Job by Job in the Job resource by specifying a level of **VirtualFull**.

A typical Job resource definition might look like the following:

```
Job {
    Name = "MyBackup"
    Type = Backup
    Client=localhost-fd
    FileSet = "Full Set"
    Storage = File
    Messages = Standard
    Pool = Default
    SpoolData = yes
}

# Default pool definition
Pool {
    Name = Default
    Pool Type = Backup
    Recycle = yes           # Automatically recycle Volumes
    AutoPrune = yes        # Prune expired volumes
    Volume Retention = 365d # one year
    NextPool = Full
    Storage = File
}

Pool {
    Name = Full
    Pool Type = Backup
    Recycle = yes           # Automatically recycle Volumes
    AutoPrune = yes        # Prune expired volumes
    Volume Retention = 365d # one year
    Storage = DiskChanger
}

# Definition of file storage device
Storage {
    Name = File
    Address = localhost
    Password = "xxx"
    Device = FileStorage
    Media Type = File
    Maximum Concurrent Jobs = 5
}

# Definition of DDS Virtual tape disk storage device
Storage {
    Name = DiskChanger
    Address = localhost # N.B. Use a fully qualified name here
    Password = "yyy"
```



```

Device = DiskChanger
Media Type = DiskChangerMedia
Maximum Concurrent Jobs = 4
Autochanger = yes
}

```

Then in bconsole or via a Run schedule, you would run the job as:

```

run job=MyBackup level=Full
run job=MyBackup level=Incremental
run job=MyBackup level=Differential
run job=MyBackup level=Incremental
run job=MyBackup level=Incremental

```

So providing there were changes between each of those jobs, you would end up with a Full backup, a Differential, which includes the first Incremental backup, then two Incremental backups. All the above jobs would be written to the **Default** pool.

To consolidate those backups into a new Full backup, you would run the following:

```

run job=MyBackup level=VirtualFull

```

And it would produce a new Full backup without using the client, and the output would be written to the **Full** Pool which uses the Diskchanger Storage.

If the Virtual Full is run, and there are no prior Jobs, the Virtual Full will fail with an error.

Note, the Start and End time of the Virtual Full backup is set to the values for the last job included in the Virtual Full (in the above example, it is an Increment). This is so that if another incremental is done, which will be based on the Virtual Full, it will backup all files from the last Job included in the Virtual Full rather than from the time the Virtual Full was actually run.

3.8 Catalog Format

Bacula 3.0 comes with some changes to the catalog format. The upgrade operation will convert the FileId field of the File table from 32 bits (max 4 billion table entries) to 64 bits (very large number of items). The conversion process can take a bit of time and will likely **DOUBLE THE SIZE** of your catalog during the conversion. Also you won't be able to run jobs during this conversion period. For example, a 3 million file catalog will take 2 minutes to upgrade on a normal machine. Please don't forget to make a valid backup of your database before executing the upgrade script. See the ReleaseNotes for additional details.

3.9 64 bit Windows Client

Unfortunately, Microsoft's implementation of Volume Shadow Copy (VSS) on their 64 bit OS versions is not compatible with a 32 bit Bacula Client. As a consequence, we are also releasing a 64 bit version of the Bacula Windows Client (win64bacula-3.0.0.exe) that does work with VSS. These binaries should only be installed on 64 bit Windows operating systems. What is important is not your hardware but whether or not you have a 64 bit version of the Windows OS.

Compared to the Win32 Bacula Client, the 64 bit release contains a few differences:

1. Before installing the Win64 Bacula Client, you must totally deinstall any prior 2.4.x Client installation using the Bacula deinstallation (see the menu item). You may want to save your .conf files first.

2. Only the Client (File daemon) is ported to Win64, the Director and the Storage daemon are not in the 64 bit Windows installer.
3. bwx-console is not yet ported.
4. bconsole is ported but it has not been tested.
5. The documentation is not included in the installer.
6. Due to Vista security restrictions imposed on a default installation of Vista, before upgrading the Client, you must manually stop any prior version of Bacula from running, otherwise the install will fail.
7. Due to Vista security restrictions imposed on a default installation of Vista, attempting to edit the conf files via the menu items will fail. You must directly edit the files with appropriate permissions. Generally double clicking on the appropriate .conf file will work providing you have sufficient permissions.
8. All Bacula files are now installed in **C:/Program Files/Bacula** except the main menu items, which are installed as before. This vastly simplifies the installation.
9. If you are running on a foreign language version of Windows, most likely **C:/Program Files** does not exist, so you should use the Custom installation and enter an appropriate location to install the files.
10. The 3.0.0 Win32 Client continues to install files in the locations used by prior versions. For the next version we will convert it to use the same installation conventions as the Win64 version.

This project was funded by Bacula Systems.

3.10 Duplicate Job Control

The new version of Bacula provides four new directives that give additional control over what Bacula does if duplicate jobs are started. A duplicate job in the sense we use it here means a second or subsequent job with the same name starts. This happens most frequently when the first job runs longer than expected because no tapes are available.

The four directives each take as an argument a **yes** or **no** value and are specified in the Job resource.

They are:

3.10.1 Allow Duplicate Jobs = <yes—no>

If this directive is enabled duplicate jobs will be run. If the directive is set to **no** (default) then only one job of a given name may run at one time, and the action that Bacula takes to ensure only one job runs is determined by the other directives (see below).

If **Allow Duplicate Jobs** is set to **no** and two jobs are present and none of the three directives given below permit cancelling a job, then the current job (the second one started) will be cancelled.

3.10.2 Allow Higher Duplicates = <yes—no>

If this directive is set to **yes** (default) the job with a higher priority (lower priority number) will be permitted to run, and the current job will be cancelled. If the priorities of the two jobs are the same, the outcome is determined by other directives (see below).

3.10.3 Cancel Queued Duplicates = <yes—no>

If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already queued to run but not yet running will be canceled. The default is **no**.

3.10.4 Cancel Running Duplicates = <yes—no>

If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already running will be canceled. The default is **no**.

3.11 TLS Authentication

In Bacula version 2.5.x and later, in addition to the normal Bacula CRAM-MD5 authentication that is used to authenticate each Bacula connection, you can specify that you want TLS Authentication as well, which will provide more secure authentication.

This new feature uses Bacula's existing TLS code (normally used for communications encryption) to do authentication. To use it, you must specify all the TLS directives normally used to enable communications encryption (TLS Enable, TLS Verify Peer, TLS Certificate, ...) and a new directive:

3.11.1 TLS Authenticate = yes

TLS Authenticate = yes

in the main daemon configuration resource (Director for the Director, Client for the File daemon, and Storage for the Storage daemon).

When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, Bacula will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two Bacula daemons will be done without encryption.

If you want to encrypt communications data, use the normal TLS directives but do not turn on **TLS Authenticate**.

3.12 bextract non-portable Win32 data

bextract has been enhanced to be able to restore non-portable Win32 data to any OS. Previous versions were unable to restore non-portable Win32 data to machines that did not have the Win32 BackupRead and BackupWrite API calls.

3.13 State File updated at Job Termination

In previous versions of Bacula, the state file, which provides a summary of previous jobs run in the **status** command output was updated only when Bacula terminated, thus if the daemon crashed, the state file might not contain all the run data. This version of the Bacula daemons updates the state file on each job termination.

3.14 MaxFullInterval = <time-interval>

The new Job resource directive **Max Full Interval** = <time-interval> can be used to specify the maximum time interval between **Full** backup jobs. When a job starts, if the time since the last Full backup is greater than the specified interval, and the job would normally be an **Incremental** or **Differential**, it will be automatically upgraded to a **Full** backup.

3.15 MaxDiffInterval = <time-interval>

The new Job resource directive **Max Diff Interval** = <time-interval> can be used to specify the maximum time interval between **Differential** backup jobs. When a job starts, if the time since the last Differential backup is greater than the specified interval, and the job would normally be an **Incremental**, it will be automatically upgraded to a **Differential** backup.

3.16 Honor No Dump Flag = <yes—no>

On FreeBSD systems, each file has a **no dump flag** that can be set by the user, and when it is set it is an indication to backup programs to not backup that particular file. This version of Bacula contains a new Options directive within a FileSet resource, which instructs Bacula to obey this flag. The new directive is:

```
Honor No Dump Flag = yes|no
```

The default value is **no**.

3.17 Exclude Dir Containing = <filename-string>

The **ExcludeDirContaining** = <filename> is a new directive that can be added to the Include section of the FileSet resource. If the specified filename (**filename-string**) is found on the Client in any directory to be backed up, the whole directory will be ignored (not backed up). For example:

```
# List of files to be backed up
FileSet {
    Name = "MyFileSet"
    Include {
        Options {
            signature = MD5
        }
        File = /home
        Exclude Dir Containing = .excludeme
    }
}
```

But in /home, there may be hundreds of directories of users and some people want to indicate that they don't want to have certain directories backed up. For example, with the above FileSet, if the user or sysadmin creates a file named **.excludeme** in specific directories, such as

```
/home/user/www/cache/.excludeme
/home/user/temp/.excludeme
```

then Bacula will not backup the two directories named:

```
/home/user/www/cache  
/home/user/temp
```

NOTE: subdirectories will not be backed up. That is, the directive applies to the two directories in question and any children (be they files, directories, etc).

3.18 Bacula Plugins

Support for shared object plugins has been implemented in the Linux, Unix and Win32 File daemons. The API will be documented separately in the Developer's Guide or in a new document. For the moment, there is a single plugin named **bpipe** that allows an external program to get control to backup and restore a file.

Plugins are also planned (partially implemented) in the Director and the Storage daemon.

3.18.1 Plugin Directory

Each daemon (DIR, FD, SD) has a new **Plugin Directory** directive that may be added to the daemon definition resource. The directory takes a quoted string argument, which is the name of the directory in which the daemon can find the Bacula plugins. If this directive is not specified, Bacula will not load any plugins. Since each plugin has a distinctive name, all the daemons can share the same plugin directory.

3.18.2 Plugin Options

The **Plugin Options** directive takes a quoted string argument (after the equal sign) and may be specified in the Job resource. The options specified will be passed to all plugins when they are run. This each plugin must know what it is looking for. The value defined in the Job resource can be modified by the user when he runs a Job via the **bconsole** command line prompts.

Note: this directive may be specified, and there is code to modify the string in the run command, but the plugin options are not yet passed to the plugin (i.e. not fully implemented).

3.18.3 Plugin Options ACL

The **Plugin Options ACL** directive may be specified in the Director's Console resource. It functions as all the other ACL commands do by permitting users running restricted consoles to specify a **Plugin Options** that overrides the one specified in the Job definition. Without this directive restricted consoles may not modify the Plugin Options.

3.18.4 Plugin = <plugin-command-string>

The **Plugin** directive is specified in the Include section of a FileSet resource where you put your **File = xxx** directives. For example:

```
FileSet {  
    Name = "MyFileSet"  
    Include {  
        Options {  
            signature = MD5  
        }  
        File = /home  
        Plugin = "bpipe:..."  
    }  
}
```

```
}  
}
```

In the above example, when the File daemon is processing the directives in the Include section, it will first backup all the files in **/home** then it will load the plugin named **bpipe** (actually bpipe-dir.so) from the Plugin Directory. The syntax and semantics of the Plugin directive require the first part of the string up to the colon (:) to be the name of the plugin. Everything after the first colon is ignored by the File daemon but is passed to the plugin. Thus the plugin writer may define the meaning of the rest of the string as he wishes.

Please see the next section for information about the **bpipe** Bacula plugin.

3.19 The bpipe Plugin

The **bpipe** plugin is provided in the directory src/plugins/fd/bpipe-fd.c of the Bacula source distribution. When the plugin is compiled and linking into the resulting dynamic shared object (DSO), it will have the name **bpipe-fd.so**.

The purpose of the plugin is to provide an interface to any system program for backup and restore. As specified above the **bpipe** plugin is specified in the Include section of your Job's FileSet resource. The full syntax of the plugin directive as interpreted by the **bpipe** plugin (each plugin is free to specify the syntax as it wishes) is:

```
Plugin = "<field1>:<field2>:<field3>:<field4>"
```

where

field1 is the name of the plugin with the trailing **-fd.so** stripped off, so in this case, we would put **bpipe** in this field.

field2 specifies the namespace, which for **bpipe** is the pseudo path and filename under which the backup will be saved. This pseudo path and filename will be seen by the user in the restore file tree. For example, if the value is **/MYSQL/regress.sql**, the data backed up by the plugin will be put under that "pseudo" path and filename. You must be careful to choose a naming convention that is unique to avoid a conflict with a path and filename that actually exists on your system.

field3 for the **bpipe** plugin specifies the "reader" program that is called by the plugin during backup to read the data. **bpipe** will call this program by doing a **popen** on it.

field4 for the **bpipe** plugin specifies the "writer" program that is called by the plugin during restore to write the data back to the filesystem.

Putting it all together, the full plugin directive line might look like the following:

```
Plugin = "bpipe:/MYSQL/regress.sql:mysqldump -f  
--opt --databases bacula:mysql"
```

The directive has been split into two lines, but within the **bacula-dir.conf** file would be written on a single line.

This causes the File daemon to call the **bpipe** plugin, which will write its data into the "pseudo" file **/MYSQL/regress.sql** by calling the program **mysqldump -f --opt --database bacula** to read the data during backup. The **mysqldump** command outputs all the data for the database named **bacula**, which will be read by the plugin and stored in the backup. During restore, the data that was backed up will be sent to the program specified in the last field, which in this case is **mysql**. When **mysql** is called, it will read

the data sent to it by the plugin then write it back to the same database from which it came (**bacula** in this case).

The **bpipe** plugin is a generic pipe program, that simply transmits the data from a specified program to Bacula for backup, and then from Bacula to a specified program for restore.

By using different command lines to **bpipe**, you can backup any kind of data (ASCII or binary) depending on the program called.

3.20 Microsoft Exchange Server 2003/2007 Plugin

3.20.1 Background

The Exchange plugin was made possible by a funded development project between Equinet Ltd – www.equinet.com (many thanks) and Bacula Systems. The code for the plugin was written by James Harper, and the Bacula core code by Kern Sibbald. All the code for this funded development has become part of the Bacula project. Thanks to everyone who made it happen.

3.20.2 Concepts

Although it is possible to backup Exchange using Bacula VSS the Exchange plugin adds a good deal of functionality, because while Bacula VSS completes a full backup (snapshot) of Exchange, it does not support Incremental or Differential backups, restoring is more complicated, and a single database restore is not possible.

Microsoft Exchange organises its storage into Storage Groups with Databases inside them. A default installation of Exchange will have a single Storage Group called 'First Storage Group', with two Databases inside it, "Mailbox Store (SERVER NAME)" and "Public Folder Store (SERVER NAME)", which hold user email and public folders respectively.

In the default configuration, Exchange logs everything that happens to log files, such that if you have a backup, and all the log files since, you can restore to the present time. Each Storage Group has its own set of log files and operates independently of any other Storage Groups. At the Storage Group level, the logging can be turned off by enabling a function called "Enable circular logging". At this time the Exchange plugin will not function if this option is enabled.

The plugin allows backing up of entire storage groups, and the restoring of entire storage groups or individual databases. Backing up and restoring at the individual mailbox or email item is not supported but can be simulated by use of the "Recovery" Storage Group (see below).

3.20.3 Installing

The Exchange plugin requires a DLL that is shipped with Microsoft Exchanger Server called **esebcli2.dll**. Assuming Exchange is installed correctly the Exchange plugin should find this automatically and run without any additional installation.

If the DLL can not be found automatically it will need to be copied into the Bacula installation directory (eg C:\Program Files\Bacula\bin). The Exchange API DLL is named esebcli2.dll and is found in C:\Program Files\Exchsrvr\bin on a default Exchange installation.

3.20.4 Backup up

To back up an Exchange server the Fileset definition must contain at least **Plugin = "exchange;/@EXCHANGE/Microsoft Information Store"** for the backup to work correctly. The 'ex-

change:' bit tells Bacula to look for the exchange plugin, the '@EXCHANGE' bit makes sure all the backed up files are prefixed with something that isn't going to share a name with something outside the plugin, and the 'Microsoft Information Store' bit is required also. It is also possible to add the name of a storage group to the "Plugin =" line, eg

Plugin = "exchange:/@EXCHANGE/Microsoft Information Store/First Storage Group"
if you want only a single storage group backed up.

Additionally, you can suffix the 'Plugin =' directive with ":notrunconfull" which will tell the plugin not to truncate the Exchange database at the end of a full backup.

An Incremental or Differential backup will backup only the database logs for each Storage Group by inspecting the "modified date" on each physical log file. Because of the way the Exchange API works, the last logfile backed up on each backup will always be backed up by the next Incremental or Differential backup too. This adds 5MB to each Incremental or Differential backup size but otherwise does not cause any problems.

By default, a normal VSS fileset containing all the drive letters will also back up the Exchange databases using VSS. This will interfere with the plugin and Exchange's shared ideas of when the last full backup was done, and may also truncate log files incorrectly. It is important, therefore, that the Exchange database files be excluded from the backup, although the folders the files are in should be included, or they will have to be recreated manually if a baremetal restore is done.

```
FileSet {
  Include {
    File = C:/Program Files/Exchsrvr/mdbdata
    Plugin = "exchange:..."
  }
  Exclude {
    File = C:/Program Files/Exchsrvr/mdbdata/E00.chk
    File = C:/Program Files/Exchsrvr/mdbdata/E00.log
    File = C:/Program Files/Exchsrvr/mdbdata/E000000F.log
    File = C:/Program Files/Exchsrvr/mdbdata/E0000010.log
    File = C:/Program Files/Exchsrvr/mdbdata/E0000011.log
    File = C:/Program Files/Exchsrvr/mdbdata/E00tmp.log
    File = C:/Program Files/Exchsrvr/mdbdata/priv1.edb
  }
}
```

The advantage of excluding the above files is that you can significantly reduce the size of your backup since all the important Exchange files will be properly saved by the Plugin.

3.20.5 Restoring

The restore operation is much the same as a normal Bacula restore, with the following provisos:

- The **Where** restore option must not be specified
- Each Database directory must be marked as a whole. You cannot just select (say) the .edb file and not the others.
- If a Storage Group is restored, the directory of the Storage Group must be marked too.
- It is possible to restore only a subset of the available log files, but they **must** be contiguous. Exchange will fail to restore correctly if a log file is missing from the sequence of log files
- Each database to be restored must be dismounted and marked as "Can be overwritten by restore"
- If an entire Storage Group is to be restored (eg all databases and logs in the Storage Group), then it is best to manually delete the database files from the server (eg C:\Program Files\Exchsrvr\mdbdata*) as Exchange can get confused by stray log files lying around.

3.20.6 Restoring to the Recovery Storage Group

The concept of the Recovery Storage Group is well documented by Microsoft <http://support.microsoft.com/kb/824126>, but to briefly summarize...

Microsoft Exchange allows the creation of an additional Storage Group called the Recovery Storage Group, which is used to restore an older copy of a database (e.g. before a mailbox was deleted) into without messing with the current live data. This is required as the Standard and Small Business Server versions of Exchange can not ordinarily have more than one Storage Group.

To create the Recovery Storage Group, drill down to the Server in Exchange System Manager, right click, and select "**New -> Recovery Storage Group...**". Accept or change the file locations and click OK. On the Recovery Storage Group, right click and select "**Add Database to Recover...**" and select the database you will be restoring.

Restore only the single database nominated as the database in the Recovery Storage Group. Exchange will redirect the restore to the Recovery Storage Group automatically. Then run the restore.

3.20.7 Restoring on Microsoft Server 2007

Apparently the **Exmerge** program no longer exists in Microsoft Server 2007, and hence you use a new procedure for recovering a single mail box. This procedure is documented by Microsoft at: <http://technet.microsoft.com/en-us/library/aa997694.aspx>, and involves using the **Restore-Mailbox** and **Get-MailboxStatistics** shell commands.

3.20.8 Caveats

This plugin is still being developed, so you should consider it currently in BETA test, and thus use in a production environment should be done only after very careful testing.

When doing a full backup, the Exchange database logs are truncated by Exchange as soon as the plugin has completed the backup. If the data never makes it to the backup medium (eg because of spooling) then the logs will still be truncated, but they will also not have been backed up. A solution to this is being worked on. You will have to schedule a new Full backup to ensure that your next backups will be usable.

The "Enable Circular Logging" option cannot be enabled or the plugin will fail.

Exchange insists that a successful Full backup must have taken place if an Incremental or Differential backup is desired, and the plugin will fail if this is not the case. If a restore is done, Exchange will require that a Full backup be done before an Incremental or Differential backup is done.

The plugin will most likely not work well if another backup application (eg NTBACKUP) is backing up the Exchange database, especially if the other backup application is truncating the log files.

The Exchange plugin has not been tested with the **Accurate** option, so we recommend either carefully testing or that you avoid this option for the current time.

The Exchange plugin is not called during processing the bconsole **estimate** command, and so anything that would be backed up by the plugin will not be added to the estimate total that is displayed.

3.21 libdbi Framework

As a general guideline, Bacula has support for a few catalog database drivers (MySQL, PostgreSQL, SQLite) coded natively by the Bacula team. With the libdbi implementation, which is a Bacula driver that uses libdbi to access the catalog, we have an open field to use many different kinds database engines following the needs of users.

The according to libdbi (<http://libdbi.sourceforge.net/>) project: libdbi implements a database-independent abstraction layer in C, similar to the DBI/DBD layer in Perl. Writing one generic set of code, programmers can leverage the power of multiple databases and multiple simultaneous database connections by using this framework.

Currently the libdbi driver in Bacula project only supports the same drivers natively coded in Bacula. However the libdbi project has support for many others database engines. You can view the list at <http://libdbi-drivers.sourceforge.net/>. In the future all those drivers can be supported by Bacula, however, they must be tested properly by the Bacula team.

Some of benefits of using libdbi are:

- The possibility to use proprietary databases engines in which your proprietary licenses prevent the Bacula team from developing the driver.
- The possibility to use the drivers written for the libdbi project.
- The possibility to use other database engines without recompiling Bacula to use them. Just change one line in bacula-dir.conf
- Abstract Database access, this is, unique point to code and profiling catalog database access.

The following drivers have been tested:

- PostgreSQL, with and without batch insert
- Mysql, with and without batch insert
- SQLite
- SQLite3

In the future, we will test and approve to use others databases engines (proprietary or not) like DB2, Oracle, Microsoft SQL.

To compile Bacula to support libdbi we need to configure the code with the `-with-dbi` and `-with-dbi-driver=[database]` `./configure` options, where [database] is the database engine to be used with Bacula (of course we can change the driver in file bacula-dir.conf, see below). We must configure the access port of the database engine with the option `-with-db-port`, because the libdbi framework doesn't know the default access port of each database.

The next phase is checking (or configuring) the bacula-dir.conf, example:

```
Catalog {
  Name = MyCatalog
  dbdriver = dbi:mysql; dbaddress = 127.0.0.1; dbport = 3306
  dbname = regress; user = regress; password = ""
}
```

The parameter **dbdriver** indicates that we will use the driver dbi with a mysql database. Currently the drivers supported by Bacula are: postgresql, mysql, sqlite, sqlite3; these are the names that may be added to string "dbi:".

The following limitations apply when Bacula is set to use the libdbi framework: - Not tested on the Win32 platform - A little performance is lost if comparing with native database driver. The reason is bound with the database driver provided by libdbi and the simple fact that one more layer of code was added.

It is important to remember, when compiling Bacula with libdbi, the following packages are needed:

- libdbi version 1.0.0, <http://libdbi.sourceforge.net/>

- libdbi-drivers 1.0.0, <http://libdbi-drivers.sourceforge.net/>

You can download them and compile them on your system or install the packages from your OS distribution.

3.22 Console Command Additions and Enhancements

3.22.1 Display Autochanger Content

The **status slots storage=<storage-name>** command displays autochanger content.

Slot	Volume Name	Status	Media Type	Pool
1	00001	Append	DiskChangerMedia	Default
2	00002	Append	DiskChangerMedia	Default
3*	00003	Append	DiskChangerMedia	Scratch
4				

If you an asterisk (*) appears after the slot number, you must run an **update slots** command to synchronize autochanger content with your catalog.

3.22.2 list joblog job=xxx or jobid=nnn

A new list command has been added that allows you to list the contents of the Job Log stored in the catalog for either a Job Name (fully qualified) or for a particular JobId. The **l**list command will include a line with the time and date of the entry.

Note for the catalog to have Job Log entries, you must have a directive such as:

```
catalog = all
```

In your Director's **Messages** resource.

3.22.3 Use separator for multiple commands

When using bconsole with readline, you can set the command separator with **@separator** command to one of those characters to write commands who require multiple input in one line.

```
!$%&'()*+,-/;:<>?[]^_`{|}~
```

3.22.4 Deleting Volumes

The delete volume bconsole command has been modified to require an asterisk (*) in front of a MediaId otherwise the value you enter is a taken to be a Volume name. This is so that users may delete numeric Volume names. The previous Bacula versions assumed that all input that started with a number was a MediaId.

This new behavior is indicated in the prompt if you read it carefully.

3.23 Bare Metal Recovery

The old bare metal recovery project is essentially dead. One of the main features of it was that it would build a recovery CD based on the kernel on your system. The problem was that every distribution has a different boot procedure and different scripts, and worse yet, the boot procedures and scripts change from one distribution to another. This meant that maintaining (keeping up with the changes) the rescue CD was too much work.

To replace it, a new bare metal recovery USB boot stick has been developed by Bacula Systems. This technology involves remastering a Ubuntu LiveCD to boot from a USB key.

Advantages:

1. Recovery can be done from within graphical environment.
2. Recovery can be done in a shell.
3. Ubuntu boots on a large number of Linux systems.
4. The process of updating the system and adding new packages is not too difficult.
5. The USB key can easily be upgraded to newer Ubuntu versions.
6. The USB key has writable partitions for modifications to the OS and for modification to your home directory.
7. You can add new files/directories to the USB key very easily.
8. You can save the environment from multiple machines on one USB key.
9. Bacula Systems is funding its ongoing development.

The disadvantages are:

1. The USB key is usable but currently under development.
2. Not everyone may be familiar with Ubuntu (no worse than using Knoppix)
3. Some older OSes cannot be booted from USB. This can be resolved by first booting a Ubuntu LiveCD then plugging in the USB key.
4. Currently the documentation is sketchy and not yet added to the main manual. See below ...

The documentation and the code can be found in the **rescue** package in the directory **linux/usb**.

3.24 Miscellaneous

3.24.1 Allow Mixed Priority = <yes—no>

This directive is only implemented in version 2.5 and later. When set to **yes** (default **no**), this job may run even if lower priority jobs are already running. This means a high priority job will not have to wait for other jobs to finish before starting. The scheduler will only mix priorities when all running jobs have this set to true.

Note that only higher priority jobs will start early. Suppose the director will allow two concurrent jobs, and that two jobs with priority 10 are running, with two more in the queue. If a job with priority 5 is added to the queue, it will be run as soon as one of the running jobs finishes. However, new priority 10 jobs will not be run until the priority 5 job has finished.

3.24.2 Bootstrap File Directive – FileRegex

FileRegex is a new command that can be added to the bootstrap (.bsr) file. The value is a regular expression. When specified, only matching filenames will be restored.

During a restore, if all File records are pruned from the catalog for a Job, normally Bacula can restore only all files saved. That is there is no way using the catalog to select individual files. With this new feature, Bacula will ask if you want to specify a Regex expression for extracting only a part of the full backup.

```
Building directory tree for JobId(s) 1,3 ...
There were no files inserted into the tree, so file selection
is not possible. Most likely your retention policy pruned the files

Do you want to restore all the files? (yes|no): no

Regex matching files to restore? (empty to abort): /tmp/regress/(bin|tests)/
Bootstrap records written to /tmp/regress/working/zog4-dir.restore.1.bsr
```

3.24.3 Bootstrap File Optimization Changes

In order to permit proper seeking on disk files, we have extended the bootstrap file format to include a **VolStartAddr** and **VolEndAddr** records. Each takes a 64 bit unsigned integer range (i.e. nnn-mmm) which defines the start address range and end address range respectively. These two directives replace the **VolStartFile**, **VolEndFile**, **VolStartBlock** and **VolEndBlock** directives. Bootstrap files containing the old directives will still work, but will not properly take advantage of proper disk seeking, and may read completely to the end of a disk volume during a restore. With the new format (automatically generated by the new Director), restores will seek properly and stop reading the volume when all the files have been restored.

3.24.4 Solaris ZFS/NFSv4 ACLs

This is an upgrade of the previous Solaris ACL backup code to the new library format, which will backup both the old POSIX(UFS) ACLs as well as the ZFS ACLs.

The new code can also restore POSIX(UFS) ACLs to a ZFS filesystem (it will translate the POSIX(UFS)) ACL into a ZFS/NFSv4 one) it can also be used to transfer from UFS to ZFS filesystems.

3.24.5 Virtual Tape Emulation

We now have a Virtual Tape emulator that allows us to run though 99.9% of the tape code but actually reading and writing to a disk file. Used with the **disk-changer** script, you can now emulate an autochanger with 10 drives and 700 slots. This feature is most useful in testing. It is enabled by using **Device Type = vtape** in the Storage daemon's Device directive. This feature is only implemented on Linux machines and should not be used for production.

3.24.6 Bat Enhancements

Bat (the Bacula Administration Tool) GUI program has been significantly enhanced and stabilized. In particular, there are new table based status commands; it can now be easily localized using Qt4 Linguist.

The Bat communications protocol has been significantly enhanced to improve GUI handling. Note, you **must** use a the bat that is distributed with the Director you are using otherwise the communications protocol will not work.

3.24.7 RunScript Enhancements

The **RunScript** resource has been enhanced to permit multiple commands per RunScript. Simply specify multiple **Command** directives in your RunScript.

```
Job {
  Name = aJob
  RunScript {
    Command = "/bin/echo test"
    Command = "/bin/echo an other test"
    Command = "/bin/echo 3 commands in the same runscript"
    RunsWhen = Before
  }
  ...
}
```

A new Client RunScript **RunsWhen** keyword of **AfterVSS** has been implemented, which runs the command after the Volume Shadow Copy has been made.

Console commands can be specified within a RunScript by using: **Console = <command>**, however, this command has not been carefully tested and debugged and is known to easily crash the Director. We would appreciate feedback. Due to the recursive nature of this command, we may remove it before the final release.

3.24.8 Status Enhancements

The bconsole **status dir** output has been enhanced to indicate Storage daemon job spooling and despooling activity.

3.24.9 Connect Timeout

The default connect timeout to the File daemon has been set to 3 minutes. Previously it was 30 minutes.

3.24.10 ftruncate for NFS Volumes

If you write to a Volume mounted by NFS (say on a local file server), in previous Bacula versions, when the Volume was recycled, it was not properly truncated because NFS does not implement ftruncate (file truncate). This is now corrected in the new version because we have written code (actually a kind user) that deletes and recreates the Volume, thus accomplishing the same thing as a truncate.

3.24.11 Support for Ubuntu

The new version of Bacula now recognizes the Ubuntu (and Kubuntu) version of Linux, and thus now provides correct autostart routines. Since Ubuntu officially supports Bacula, you can also obtain any recent release of Bacula from the Ubuntu repositories.

3.24.12 Recycle Pool = <pool-name>

The new **RecyclePool** directive defines to which pool the Volume will be placed (moved) when it is recycled. Without this directive, a Volume will remain in the same pool when it is recycled. With this directive, it can be moved automatically to any existing pool during a recycle. This directive is probably most useful when defined in the Scratch pool, so that volumes will be recycled back into the Scratch pool.

3.24.13 FD Version

The File daemon to Director protocol now includes a version number, which although there is no visible change for users, will help us in future versions automatically determine if a File daemon is not compatible.

3.24.14 Max Run Sched Time = <time-period-in-seconds>

The time specifies the maximum allowed time that a job may run, counted from when the job was scheduled. This can be useful to prevent jobs from running during working hours. We can see it like **Max Start Delay + Max Run Time**.

3.24.15 Max Wait Time = <time-period-in-seconds>

Previous **MaxWaitTime** directives aren't working as expected, instead of checking the maximum allowed time that a job may block for a resource, those directives worked like **MaxRunTime**. Some users are reporting to use **Incr/Diff/Full Max Wait Time** to control the maximum run time of their job depending on the level. Now, they have to use **Incr/Diff/Full Max Run Time**. **Incr/Diff/Full Max Wait Time** directives are now deprecated.

3.24.16 Incremental—Differential Max Wait Time = <time-period-in-seconds>

These directives have been deprecated in favor of **Incremental|Differential Max Run Time**.

3.24.17 Max Run Time directives

Using **Full/Diff/Incr Max Run Time**, it's now possible to specify the maximum allowed time that a job can run depending on the level.



3.24.18 Statistics Enhancements

If you (or probably your boss) want to have statistics on your backups to provide some *Service Level Agreement* indicators, you could use a few SQL queries on the Job table to report how many:

- jobs have run
- jobs have been successful
- files have been backed up
- ...

However, these statistics are accurate only if your job retention is greater than your statistics period. Ie, if jobs are purged from the catalog, you won't be able to use them.

Now, you can use the **update stats** [**days=num**] console command to fill the JobHistory table with new Job records. If you want to be sure to take in account only **good jobs**, ie if one of your important job has failed but you have fixed the problem and restarted it on time, you probably want to delete the first *bad* job record and keep only the successful one. For that simply let your staff do the job, and update JobHistory table after two or three days depending on your organization using the [**days=num**] option.

These statistics records aren't used for restoring, but mainly for capacity planning, billings, etc.

The Bweb interface provides a statistics module that can use this feature. You can also use tools like Talend or extract information by yourself.

The **Statistics Retention = <time>** director directive defines the length of time that Bacula will keep statistics job records in the Catalog database after the Job End time. (In **JobHistory** table) When this time period expires, and if user runs **prune stats** command, Bacula will prune (remove) Job records that are older than the specified period.

You can use the following Job resource in your nightly **BackupCatalog** job to maintain statistics.

```
Job {
  Name = BackupCatalog
  ...
  RunScript {
    Console = "update stats days=3"
    Console = "prune stats yes"
    RunsWhen = After
    RunsOnClient = no
  }
}
```

3.24.19 ScratchPool = <pool-resource-name>

This directive permits to specify a specific *Scratch* pool for the current pool. This is useful when using multiple storage sharing the same mediatype or when you want to dedicate volumes to a particular set of pool.

3.24.20 Enhanced Attribute Despooling

If the storage daemon and the Director are on the same machine, the spool file that contains attributes is read directly by the Director instead of being transmitted across the network. That should reduce load and speedup insertion.

3.24.21 SpoolSize = <size-specification-in-bytes>

A new Job directive permits to specify the spool size per job. This is used in advanced job tuning. **SpoolSize=bytes**

3.24.22 MaxConsoleConnections = <number>

A new director directive permits to specify the maximum number of Console Connections that could run concurrently. The default is set to 20, but you may set it to a larger number.

3.24.23 VerId = <string>

A new director directive permits to specify a personal identifier that will be displayed in the **version** command.

3.24.24 dbcheck enhancements

If you are using Mysql, dbcheck will now ask you if you want to create temporary indexes to speed up orphaned Path and Filename elimination.

A new **-B** option allows you to print catalog information in a simple text based format. This is useful to backup it in a secure way.

```
$ dbcheck -B
catalog=MyCatalog
db_type=SQLite
db_name=regress
db_driver=
db_user=regress
db_password=
db_address=
db_port=0
db_socket=
```

You can now specify the database connection port in the command line.

3.24.25 --docdir configure option

You can use **--docdir=** on the **./configure** command to specify the directory where you want Bacula to install the LICENSE, ReleaseNotes, ChangeLog, ... files. The default is **/usr/share/doc/bacula**.

3.24.26 --htmldir configure option

You can use **--htmldir=** on the **./configure** command to specify the directory where you want Bacula to install the bat html help files. The default is **/usr/share/bacula/html**

3.24.27 --with-pluginindir configure option

You can use **--pluginindir=** on the **./configure** command to specify the directory where you want Bacula to install the plugins (currently only bpipe-fd). The default is **/usr/lib**.

Chapter 4

Bacula FD Plugin API

To write a Bacula plugin, you create a dynamic shared object program (or dll on Win32) with a particular name and two exported entry points, place it in the **Plugins Directory**, which is defined in the **bacula-fd.conf** file in the **Client** resource, and when the FD starts, it will load all the plugins that end with **-fd.so** (or **-fd.dll** on Win32) found in that directory.

4.1 Normal vs Command Plugins

In general, there are two ways that plugins are called. The first way, is when a particular event is detected in Bacula, it will transfer control to each plugin that is loaded in turn informing the plugin of the event. This is very similar to how a **RunScript** works, and the events are very similar. Once the plugin gets control, it can interact with Bacula by getting and setting Bacula variables. In this way, it behaves much like a RunScript. Currently very few Bacula variables are defined, but they will be implemented as the need arises, and it is very extensible.

We plan to have plugins register to receive events that they normally would not receive, such as an event for each file examined for backup or restore. This feature is not yet implemented.

The second type of plugin, which is more useful and fully implemented in the current version is what we call a command plugin. As with all plugins, it gets notified of important events as noted above (details described below), but in addition, this kind of plugin can accept a command line, which is a:

```
Plugin = <command-string>
```

directive that is placed in the Include section of a FileSet and is very similar to the "File = " directive. When this Plugin directive is encountered by Bacula during backup, it passes the "command" part of the Plugin directive only to the plugin that is explicitly named in the first field of that command string. This allows that plugin to backup any file or files on the system that it wants. It can even create "virtual files" in the catalog that contain data to be restored but do not necessarily correspond to actual files on the filesystem.

The important features of the command plugin entry points are:

- It is triggered by a "Plugin =" directive in the FileSet
- Only a single plugin is called that is named on the "Plugin =" directive.
- The full command string after the "Plugin =" is passed to the plugin so that it can be told what to backup/restore.

4.2 Loading Plugins

Once the File daemon loads the plugins, it asks the OS for the two entry points (loadPlugin and unloadPlugin) then calls the **loadPlugin** entry point (see below).

Bacula passes information to the plugin through this call and it gets back information that it needs to use the plugin. Later, Bacula will call particular functions that are defined by the **loadPlugin** interface.

When Bacula is finished with the plugin (when Bacula is going to exit), it will call the **unloadPlugin** entry point.

The two entry points are:

```
bRC loadPlugin(bInfo *lbinfo, bFuncs *lbfuncs, pInfo **pinfo, pFuncs **pfuncs)
```

and

```
bRC unloadPlugin()
```

both these external entry points to the shared object are defined as C entry points to avoid name mangling complications with C++. However, the shared object can actually be written in any language (preferably C or C++) providing that it follows C language calling conventions.

The definitions for **bRC** and the arguments are **src/typed/fd-plugins.h** and so this header file needs to be included in your plugin. It along with **src/lib/plugins.h** define basically the whole plugin interface. Within this header file, it includes the following files:

```
#include <sys/types.h>
#include "config.h"
#include "bc_types.h"
#include "lib/plugins.h"
#include <sys/stat.h>
```

Aside from the **bc_types.h** and **config.h** headers, the plugin definition uses the minimum code from Bacula. The **bc_types.h** file is required to ensure that the data type definitions in arguments correspond to the Bacula core code.

The return codes are defined as:

```
typedef enum {
    bRC_OK      = 0,          /* OK */
    bRC_Stop    = 1,          /* Stop calling other plugins */
    bRC_Error   = 2,          /* Some kind of error */
    bRC_More    = 3,          /* More files to backup */
} bRC;
```

At a future point in time, we hope to make the Bacula libbac.a into a shared object so that the plugin can use much more of Bacula's infrastructure, but for this first cut, we have tried to minimize the dependence on Bacula.

4.3 loadPlugin

As previously mentioned, the **loadPlugin** entry point in the plugin is called immediately after Bacula loads the plugin when the File daemon itself is first starting. This entry point is only called once during the execution of the File daemon. In calling the plugin, the first two arguments are information from Bacula

that is passed to the plugin, and the last two arguments are information about the plugin that the plugin must return to Bacula. The call is:

```
bRC loadPlugin(bInfo *lbinfo, bFuncs *lbfuncs, pInfo **pinfo, pFuncs **pfuncs)
```

and the arguments are:

lbinfo This is information about Bacula in general. Currently, the only value defined in the bInfo structure is the version, which is the Bacula plugin interface version, currently defined as 1. The **size** is set to the byte size of the structure. The exact definition of the bInfo structure as of this writing is:

```
typedef struct s_baculaInfo {
    uint32_t size;
    uint32_t version;
} bInfo;
```

lbfuncs The bFuncs structure defines the callback entry points within Bacula that the plugin can use register events, get Bacula values, set Bacula values, and send messages to the Job output or debug output. The exact definition as of this writing is:

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

We will discuss these entry points and how to use them a bit later when describing the plugin code.

pInfo When the loadPlugin entry point is called, the plugin must initialize an information structure about the plugin and return a pointer to this structure to Bacula.

The exact definition as of this writing is:

```
typedef struct s_pluginInfo {
    uint32_t size;
    uint32_t version;
    const char *plugin_magic;
    const char *plugin_license;
    const char *plugin_author;
    const char *plugin_date;
    const char *plugin_version;
    const char *plugin_description;
} pInfo;
```

Where:

version is the current Bacula defined plugin interface version, currently set to 1. If the interface version differs from the current version of Bacula, the plugin will not be run (not yet implemented).

plugin_magic is a pointer to the text string `"*FDPluginData*"`, a sort of sanity check. If this value is not specified, the plugin will not be run (not yet implemented).

plugin_license is a pointer to a text string that describes the plugin license. Bacula will only accept compatible licenses (not yet implemented).

plugin_author is a pointer to the text name of the author of the program. This string can be anything but is generally the author's name.

plugin_date is the pointer text string containing the date of the plugin. This string can be anything but is generally some human readable form of the date.

plugin_version is a pointer to a text string containing the version of the plugin. The contents are determined by the plugin writer.

plugin_description is a pointer to a string describing what the plugin does. The contents are determined by the plugin writer.

The pInfo structure must be defined in static memory because Bacula does not copy it and may refer to the values at any time while the plugin is loaded. All values must be supplied or the plugin will not run (not yet implemented). All text strings must be either ASCII or UTF-8 strings that are terminated with a zero byte.

pFuncs When the loadPlugin entry point is called, the plugin must initialize an entry point structure about the plugin and return a pointer to this structure to Bacula. This structure contains pointer to each of the entry points that the plugin must provide for Bacula. When Bacula is actually running the plugin, it will call the defined entry points at particular times. All entry points must be defined.

The pFuncs structure must be defined in static memory because Bacula does not copy it and may refer to the values at any time while the plugin is loaded.

The exact definition as of this writing is:

```
typedef struct s_pluginFuncs {
    uint32_t size;
    uint32_t version;
    bRC (*newPlugin)(bpContext *ctx);
    bRC (*freePlugin)(bpContext *ctx);
    bRC (*getPluginValue)(bpContext *ctx, pVariable var, void *value);
    bRC (*setPluginValue)(bpContext *ctx, pVariable var, void *value);
    bRC (*handlePluginEvent)(bpContext *ctx, bEvent *event, void *value);
    bRC (*startBackupFile)(bpContext *ctx, struct save_pkt *sp);
    bRC (*endBackupFile)(bpContext *ctx);
    bRC (*startRestoreFile)(bpContext *ctx, const char *cmd);
    bRC (*endRestoreFile)(bpContext *ctx);
    bRC (*pluginIO)(bpContext *ctx, struct io_pkt *io);
    bRC (*createFile)(bpContext *ctx, struct restore_pkt *rp);
    bRC (*setFileAttributes)(bpContext *ctx, struct restore_pkt *rp);
    bRC (*checkFile)(bpContext *ctx, char *fname);
} pFuncs;
```

The details of the entry points will be presented in separate sections below.

Where:

size is the byte size of the structure.

version is the plugin interface version currently set to 3.

Sample code for loadPlugin:

```
bfuns = lbfuns;           /* set Bacula funct pointers */
binfo = lbinfo;
*pinfo = &pluginInfo;     /* return pointer to our info */
*pfuns = &pluginFuncs;    /* return pointer to our functions */

return bRC_OK;
```

where pluginInfo and pluginFuncs are statically defined structures. See bpipe-fd.c for details.

4.4 Plugin Entry Points

This section will describe each of the entry points (subroutines) within the plugin that the plugin must provide for Bacula, when they are called and their arguments. As noted above, pointers to these subroutines are passed back to Bacula in the pFuncs structure when Bacula calls the loadPlugin() externally defined entry point.

4.4.1 newPlugin(bpContext *ctx)

This is the entry point that Bacula will call when a new "instance" of the plugin is created. This typically happens at the beginning of a Job. If 10 Jobs are running simultaneously, there will be at least 10 instances of the plugin.

The bpContext structure will be passed to the plugin, and during this call, if the plugin needs to have its private working storage that is associated with the particular instance of the plugin, it should create it from the heap (malloc the memory) and store a pointer to its private working storage in the **pContext** variable. Note: since Bacula is a multi-threaded program, you must not keep any variable data in your plugin unless it is truly meant to apply globally to the whole plugin. In addition, you must be aware that except the first and last call to the plugin (loadPlugin and unloadPlugin) all the other calls will be made by threads that correspond to a Bacula job. The bpContext that will be passed for each thread will remain the same throughout the Job thus you can keep your private Job specific data in it (**bContext**).

```
typedef struct s_bpContext {
    void *pContext;    /* Plugin private context */
    void *bContext;    /* Bacula private context */
} bpContext;
```

This context pointer will be passed as the first argument to all the entry points that Bacula calls within the plugin. Needless to say, the plugin should not change the bContext variable, which is Bacula's private context pointer for this instance (Job) of this plugin.

4.4.2 freePlugin(bpContext *ctx)

This entry point is called when this instance of the plugin is no longer needed (the Job is ending), and the plugin should release all memory it may have allocated for this particular instance (Job) i.e. the pContext. This is not the final termination of the plugin signaled by a call to **unloadPlugin**. Any other instances (Job) will continue to run, and the entry point **newPlugin** may be called again if other jobs start.

4.4.3 getPluginValue(bpContext *ctx, pVariable var, void *value)

Bacula will call this entry point to get a value from the plugin. This entry point is currently not called.

4.4.4 setPluginValue(bpContext *ctx, pVariable var, void *value)

Bacula will call this entry point to set a value in the plugin. This entry point is currently not called.

4.4.5 handlePluginEvent(bpContext *ctx, bEvent *event, void *value)

This entry point is called when Bacula encounters certain events (discussed below). This is, in fact, the main way that most plugins get control when a Job runs and how they know what is happening in the job.

It can be likened to the **RunScript** feature that calls external programs and scripts, and is very similar to the Bacula Python interface. When the plugin is called, Bacula passes it the pointer to an event structure (bEvent), which currently has one item, the eventType:

```
typedef struct s_bEvent {
    uint32_t eventType;
} bEvent;
```

which defines what event has been triggered, and for each event, Bacula will pass a pointer to a value associated with that event. If no value is associated with a particular event, Bacula will pass a NULL pointer, so the plugin must be careful to always check value pointer prior to dereferencing it.

The current list of events are:

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

Most of the above are self-explanatory.

bEventJobStart is called whenever a Job starts. The value passed is a pointer to a string that contains: "Jobid=nnn Job=job-name". Where nnn will be replaced by the JobId and job-name will be replaced by the Job name. The variable is temporary so if you need the values, you must copy them.

bEventJobEnd is called whenever a Job ends. No value is passed.

bEventStartBackupJob is called when a Backup Job begins. No value is passed.

bEventEndBackupJob is called when a Backup Job ends. No value is passed.

bEventStartRestoreJob is called when a Restore Job starts. No value is passed.

bEventEndRestoreJob is called when a Restore Job ends. No value is passed.

bEventStartVerifyJob is called when a Verify Job starts. No value is passed.

bEventEndVerifyJob is called when a Verify Job ends. No value is passed.

bEventBackupCommand is called prior to the bEventStartBackupJob and the plugin is passed the command string (everything after the equal sign in "Plugin =" as the value.

Note, if you intend to backup a file, this is an important first point to write code that copies the command string passed into your pContext area so that you will know that a backup is being performed and you will know the full contents of the "Plugin =" command (i.e. what to backup and what virtual filename the user wants to call it.

bEventRestoreCommand is called prior to the bEventStartRestoreJob and the plugin is passed the command string (everything after the equal sign in "Plugin =" as the value.

See the notes above concerning backup and the command string. This is the point at which Bacula passes you the original command string that was specified during the backup, so you will want to save it in your pContext area for later use when Bacula calls the plugin again.

bEventLevel is called when the level is set for a new Job. The value is a 32 bit integer stored in the void*, which represents the Job Level code.

bEventSince is called when the since time is set for a new Job. The value is a time_t time at which the last job was run.

During each of the above calls, the plugin receives either no specific value or only one value, which in some cases may not be sufficient. However, knowing the context of the event, the plugin can call back to the Bacula entry points it was passed during the **loadPlugin** call and get to a number of Bacula variables. (at the current time few Bacula variables are implemented, but it easily extended at a future time and as needs require).

4.4.6 startBackupFile(bpContext *ctx, struct save_pkt *sp)

This entry point is called only if your plugin is a command plugin, and it is called when Bacula encounters the "Plugin = " directive in the Include section of the FileSet. Called when beginning the backup of a file. Here Bacula provides you with a pointer to the **save_pkt** structure and you must fill in this packet with the "attribute" data of the file.

```
struct save_pkt {
    int32_t pkt_size;           /* size of this packet */
    char *fname;                /* Full path and filename */
    char *link;                 /* Link name if any */
    struct stat statp;          /* System stat() packet for file */
    int32_t type;               /* FT_xx for this file */
    uint32_t flags;             /* Bacula internal flags */
    bool portable;              /* set if data format is portable */
    char *cmd;                  /* command */
    int32_t pkt_end;            /* end packet sentinel */
};
```

The second argument is a pointer to the **save_pkt** structure for the file to be backed up. The plugin is responsible for filling in all the fields of the **save_pkt**. If you are backing up a real file, then generally, the statp structure can be filled in by doing a **stat** system call on the file.

If you are backing up a database or something that is more complex, you might want to create a virtual file. That is a file that does not actually exist on the filesystem, but represents say an object that you are backing up. In that case, you need to ensure that the **fname** string that you pass back is unique so that it does not conflict with a real file on the system, and you need to artificially create values in the statp packet.

Example programs such as **bpipe-fd.c** show how to set these fields. You must take care not to store pointers the stack in the pointer fields such as fname and link, because when you return from your function, your stack entries will be destroyed. The solution in that case is to malloc() and return the pointer to it. In order to not have memory leaks, you should store a pointer to all memory allocated in your pContext structure so that in subsequent calls or at termination, you can release it back to the system.

Once the backup has begun, Bacula will call your plugin at the **pluginIO** entry point to "read" the data to be backed up. Please see the **bpipe-fd.c** plugin for how to do I/O.

Example of filling in the save_pkt as used in bpipe-fd.c:

```
struct plugin_ctx *p_ctx = (struct plugin_ctx *)ctx->pContext;
time_t now = time(NULL);
sp->fname = p_ctx->fname;
sp->statp.st_mode = 0700 | S_IFREG;
sp->statp.st_ctime = now;
sp->statp.st_mtime = now;
```

```

sp->statp.st_atime = now;
sp->statp.st_size = -1;
sp->statp.st_blksize = 4096;
sp->statp.st_blocks = 1;
p_ctx->backup = true;
return bRC_OK;

```

Note: the filename to be created has already been created from the command string previously sent to the plugin and is in the plugin context (p.ctx->fname) and is a malloc()ed string. This example creates a regular file (S_IFREG), with various fields being created.

In general, the sequence of commands issued from Bacula to the plugin to do a backup while processing the "Plugin = " directive are:

1. generate a bEventBackupCommand event to the specified plugin and pass it the command string.
2. make a startPluginBackup call to the plugin, which fills in the data needed in save_pkt to save as the file attributes and to put on the Volume and in the catalog.
3. call Bacula's internal save_file() subroutine to save the specified file. The plugin will then be called at pluginIO() to "open" the file, and then to read the file data. Note, if you are dealing with a virtual file, the "open" operation is something the plugin does internally and it doesn't necessarily mean opening a file on the filesystem. For example in the case of the bpipe-fd.c program, it initiates a pipe to the requested program. Finally when the plugin signals to Bacula that all the data was read, Bacula will call the plugin with the "close" pluginIO() function.

4.4.7 endBackupFile(bpContext *ctx)

Called at the end of backing up a file for a command plugin. If the plugin's work is done, it should return bRC_OK. If the plugin wishes to create another file and back it up, then it must return bRC_More (not yet implemented). This is probably a good time to release any malloc()ed memory you used to pass back filenames.

4.4.8 startRestoreFile(bpContext *ctx, const char *cmd)

Called when the first record is read from the Volume that was previously written by the command plugin.

4.4.9 createFile(bpContext *ctx, struct restore_pkt *rp)

Called for a command plugin to create a file during a Restore job before restoring the data. This entry point is called before any I/O is done on the file. After this call, Bacula will call pluginIO() to open the file for write.

The data in the restore_pkt is passed to the plugin and is based on the data that was originally given by the plugin during the backup and the current user restore settings (e.g. where, RegexWhere, replace). This allows the plugin to first create a file (if necessary) so that the data can be transmitted to it. The next call to the plugin will be a pluginIO command with a request to open the file write-only.

This call must return one of the following values:

```

enum {
    CF_SKIP = 1,          /* skip file (not newer or something) */
    CF_ERROR,             /* error creating file */
    CF_EXTRACT,           /* file created, data to extract */
    CF_CREATED            /* file created, no data to extract */
};

```

in the `restore_pkt` value `create_status`. For a normal file, unless there is an error, you must return `CF_EXTRACT`.

```
struct restore_pkt {
    int32_t pkt_size;           /* size of this packet */
    int32_t stream;             /* attribute stream id */
    int32_t data_stream;        /* id of data stream to follow */
    int32_t type;               /* file type FT */
    int32_t file_index;         /* file index */
    int32_t LinkFI;             /* file index to data if hard link */
    uid_t uid;                  /* userid */
    struct stat statp;          /* decoded stat packet */
    const char *attrEx;         /* extended attributes if any */
    const char *ofname;         /* output filename */
    const char *olname;         /* output link name */
    const char *where;          /* where */
    const char *RegexWhere;     /* regex where */
    int replace;                /* replace flag */
    int create_status;          /* status from createFile() */
    int32_t pkt_end;            /* end packet sentinel */
};
```

Typical code to create a regular file would be the following:

```
struct plugin_ctx *p_ctx = (struct plugin_ctx *)ctx->pContext;
time_t now = time(NULL);
sp->fname = p_ctx->fname; /* set the full path/filename I want to create */
sp->type = FT_REG;
sp->statp.st_mode = 0700 | S_IFREG;
sp->statp.st_ctime = now;
sp->statp.st_mtime = now;
sp->statp.st_atime = now;
sp->statp.st_size = -1;
sp->statp.st_blksize = 4096;
sp->statp.st_blocks = 1;
return bRC_OK;
```

This will create a virtual file. If you are creating a file that actually exists, you will most likely want to fill the `statp` packet using the `stat()` system call.

Creating a directory is similar, but requires a few extra steps:

```
struct plugin_ctx *p_ctx = (struct plugin_ctx *)ctx->pContext;
time_t now = time(NULL);
sp->fname = p_ctx->fname; /* set the full path I want to create */
sp->link = xxx; where xxx is p_ctx->fname with a trailing forward slash
sp->type = FT_DIREND
sp->statp.st_mode = 0700 | S_IFDIR;
sp->statp.st_ctime = now;
sp->statp.st_mtime = now;
sp->statp.st_atime = now;
sp->statp.st_size = -1;
sp->statp.st_blksize = 4096;
sp->statp.st_blocks = 1;
return bRC_OK;
```

The link field must be set with the full cononical path name, which always ends with a forward slash. If you do not terminate it with a forward slash, you will surely have problems later.

As with the example that creates a file, if you are backing up a real directory, you will want to do an stat() on the directory.

Note, if you want the directory permissions and times to be correctly restored, you must create the directory **after** all the file directories have been sent to Bacula. That allows the restore process to restore all the files in a directory using default directory options, then at the end, restore the directory permissions. If you do it the other way around, each time you restore a file, the OS will modify the time values for the directory entry.

4.4.10 setFileAttributes(bpContext *ctx, struct restore_pkt *rp)

This is call not yet implemented. Called for a command plugin.

See the definition of **restre_pkt** in the above section.

4.4.11 endRestoreFile(bpContext *ctx)

Called when a command plugin is done restoring a file.

4.4.12 pluginIO(bpContext *ctx, struct io_pkt *io)

Called to do the input (backup) or output (restore) of data from or to a file for a command plugin. These routines simulate the Unix read(), write(), open(), close(), and lseek() I/O calls, and the arguments are passed in the packet and the return values are also placed in the packet. In addition for Win32 systems the plugin must return two additional values (described below).

```
enum {
    IO_OPEN = 1,
    IO_READ = 2,
    IO_WRITE = 3,
    IO_CLOSE = 4,
    IO_SEEK = 5
};

struct io_pkt {
    int32_t pkt_size;           /* Size of this packet */
    int32_t func;               /* Function code */
    int32_t count;              /* read/write count */
    mode_t mode;                /* permissions for created files */
    int32_t flags;              /* Open flags */
    char *buf;                  /* read/write buffer */
    const char *fname;          /* open filename */
    int32_t status;             /* return status */
    int32_t io_errno;           /* errno code */
    int32_t lerror;             /* Win32 error code */
    int32_t whence;             /* lseek argument */
    boffset_t offset;           /* lseek argument */
    bool win32;                 /* Win32 GetLastError returned */
    int32_t pkt_end;            /* end packet sentinel */
};
```

The particular Unix function being simulated is indicated by the **func**, which will have one of the IO_OPEN, IO_READ, ... codes listed above. The status code that would be returned from a Unix call is returned in **status** for IO_OPEN, IO_CLOSE, IO_READ, and IO_WRITE. The return value for IO_SEEK is returned in **offset** which in general is a 64 bit value.

When there is an error on Unix systems, you must always set **io_error**, and on a Win32 system, you must always set **win32**, and the returned value from the OS call GetLastError() in **lerror**.

For all except IO_SEEK, **status** is the return result. In general it is a positive integer unless there is an error in which case it is -1.

The following describes each call and what you get and what you should return:

IO_OPEN You will be passed **fname**, **mode**, and **flags**. You must set on return: **status**, and if there is a Unix error **io_errno** must be set to the **errno** value, and if there is a Win32 error **win32** and **lerror**.

IO_READ You will be passed: **count**, and **buf** (buffer of size **count**). You must set on return: **status** to the number of bytes read into the buffer (**buf**) or -1 on an error, and if there is a Unix error **io_errno** must be set to the **errno** value, and if there is a Win32 error, **win32** and **lerror** must be set.

IO_WRITE You will be passed: **count**, and **buf** (buffer of size **count**). You must set on return: **status** to the number of bytes written from the buffer (**buf**) or -1 on an error, and if there is a Unix error **io_errno** must be set to the **errno** value, and if there is a Win32 error, **win32** and **lerror** must be set.

IO_CLOSE Nothing will be passed to you. On return you must set **status** to 0 on success and -1 on failure. If there is a Unix error **io_errno** must be set to the **errno** value, and if there is a Win32 error, **win32** and **lerror** must be set.

IO_LSEEK You will be passed: **offset**, and **whence**. **offset** is a 64 bit value and is the position to seek to relative to **whence**. **whence** is one of the following SEEK_SET, SEEK_CUR, or SEEK_END indicating to either to seek to an absolute position, relative to the current position or relative to the end of the file. You must pass back in **offset** the absolute location to which you seeked. If there is an error, **offset** should be set to -1. If there is a Unix error **io_errno** must be set to the **errno** value, and if there is a Win32 error, **win32** and **lerror** must be set.

Note: Bacula will call IO_SEEK only when writing a sparse file.

4.4.13 bool checkFile(bpContext *ctx, char *fname)

If this entry point is set, Bacula will call it after backing up all file data during an Accurate backup. It will be passed the full filename for each file that Bacula is proposing to mark as deleted. Only files previously backed up but not backed up in the current session will be marked to be deleted. If you return **false**, the file will be marked deleted. If you return **true** the file will not be marked deleted. This permits a plugin to ensure that previously saved virtual files or files controlled by your plugin that have not change (not backed up in the current job) are not marked to be deleted. This entry point will only be called during Accurate Incremental and Differential backup jobs.

4.5 Bacula Plugin Entrypoints

When Bacula calls one of your plugin entrypoints, you can call back to the entrypoints in Bacula that were supplied during the xxx plugin call to get or set information within Bacula.

4.5.1 bRC registerBaculaEvents(bpContext *ctx, ...)

This Bacula entrypoint will allow you to register to receive events that are not automatically passed to your plugin by default. This entrypoint currently is unimplemented.

4.5.2 bRC getBaculaValue(bpContext *ctx, bVariable var, void *value)

Calling this entrypoint, you can obtain specific values that are available in Bacula. The following Variables can be referenced:

- bVarJobId returns an int
- bVarFDName returns a char *
- bVarLevel returns an int
- bVarClient returns a char *
- bVarJobName returns a char *
- bVarJobStatus returns an int
- bVarSinceTime returns an int (time_t)
- bVarAccurate returns an int

4.5.3 bRC setBaculaValue(bpContext *ctx, bVariable var, void *value)

Calling this entrypoint allows you to set particular values in Bacula. The only variable that can currently be set is **bVarFileSeen** and the value passed is a char * that points to the full filename for a file that you are indicating has been seen and hence is not deleted.

4.5.4 bRC JobMessage(bpContext *ctx, const char *file, int line, int type, utime_t mtime, const char *fmt, ...)

This call permits you to put a message in the Job Report.

4.5.5 bRC DebugMessage(bpContext *ctx, const char *file, int line, int level, const char *fmt, ...)

This call permits you to print a debug message.

4.5.6 void baculaMalloc(bpContext *ctx, const char *file, int line, size_t size)

This call permits you to obtain memory from Bacula's memory allocator.

4.5.7 void baculaFree(bpContext *ctx, const char *file, int line, void *mem)

This call permits you to free memory obtained from Bacula's memory allocator.

4.6 Building Bacula Plugins

There is currently one sample program **example-plugin-fd.c** and one working plugin **bpipe-fd.c** that can be found in the Bacula **src/plugins/fd** directory. Both are built with the following:

```
cd <bacula-source>
./configure <your-options>
make
...
cd src/plugins/fd
make
make test
```

After building Bacula and changing into the `src/plugins/fd` directory, the **make** command will build the **bpipe-fd.so** plugin, which is a very useful and working program.

The **make test** command will build the **example-plugin-fd.so** plugin and a binary named **main**, which is build from the source code located in **src/filed/fd_plugins.c**.

If you execute `./main`, it will load and run the example-plugin-fd plugin simulating a small number of the calling sequences that Bacula uses in calling a real plugin. This allows you to do initial testing of your plugin prior to trying it with Bacula.

You can get a good idea of how to write your own plugin by first studying the example-plugin-fd, and actually running it. Then it can also be instructive to read the `bpipe-fd.c` code as it is a real plugin, which is still rather simple and small.

When actually writing your own plugin, you may use the `example-plugin-fd.c` code as a template for your code.

Chapter 5

The Current State of Bacula

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

5.1 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.

5.2 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.

5.3 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).

5.4 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 6

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 7

Supported Operating Systems

X Fully supported

★ They are reported to work in many cases. However they are NOT supported by the bacula's project.

Operating Systems	Version	Client Daemon	Director Daemon	Storage Daemon
GNU/Linux	All	X	X	X
FreeBSD	≥ 5.0	X	X	X
Solaris	≥ 8	X	X	X
OpenSolaris		X	X	X
MS Windows 32bit	Win98/Me	X		
	WinNT/2K	X	★	★
	XP	X	★	★
	2008/Vista	X	★	★
MS Windows 64bit	2008/Vista	X		
MacOS X/Darwin		X		
OpenBSD		X	★	
NetBSD		X	★	
Irix		★		
True64		★		
AIX	≥ 4.3	★		
BSDI		★		
HPUX		★		

Important notes

- By GNU/Linux, we mean 32/64bit Gentoo, Red Hat, Fedora, Mandriva, Debian, OpenSuSE, Ubuntu, Kubuntu, ...
- For FreeBSD older than version 5.0, please see some **important** considerations in the Tape Modes on FreeBSD section of the Tape Testing chapter of this manual.
- MS Windows Director and Storage daemon are available in the binary Client installer
- For MacOSX see <http://fink.sourceforge.net/> for obtaining the packages

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 8

Supported Tape Drives

Bacula uses standard operating system calls (read, write, ioctl) to interface to tape drives. As a consequence, it relies on having a correctly written OS tape driver. Bacula is known to work perfectly well with SCSI tape drivers on FreeBSD, Linux, Solaris, and Windows machines, and it may work on other *nix machines, 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	Capacity
-	ADIC	DLT	Adic Scalar 100 DLT	100GB
-	ADIC	DLT	Adic Fastor 22 DLT	-
FreeBSD 5.4-RELEASE-p1 amd64	Certance	LTO	AdicCertance CL400 LTO Ultrium 2	200GB
-	-	DDS	Compaq DDS 2,3,4	-
SuSE 8.1 Pro	Compaq	AIT	Compaq AIT 35 LVD	35/70GB
-	Exabyte	-	Exabyte drives less than 10 years old	-
-	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 and 3590 drives	-
FreeBSD 4.10 RELEASE	HP	DAT	HP StorageWorks DAT72i	-
-	Overland	LTO	LoaderXpress LTO	-
-	Overland	-	Neo2000	-
-	OnStream	-	OnStream drives (see below)	-
FreeBSD 4.11-Release	Quantum	SDLT	SDLT320	160/320GB
-	Quantum	DLT	DLT-8000	40/80GB
Linux	Seagate	DDS-4	Scorpio 40	20/40GB

FreeBSD 4.9 STABLE	Seagate	DDS-4	STA2401LW	20/40GB
FreeBSD 5.2.1 pthreads patched RELEASE	Seagate	AIT-1	STA1701W	35/70GB
Linux	Sony	DDS-2,3,4	-	4-40GB
Linux	Tandberg	-	Tandbert MLR3	-
FreeBSD	Tandberg	-	Tandberg SLR6	-
Solaris	Tandberg	-	Tandberg SLR75	-

There is a list of supported autochangers in the Supported Autochangers chapter of this document, where you will find other tape drives that work with Bacula.

8.1 Unsupported Tape Drives

Previously OnStream IDE-SCSI tape drives did not work with Bacula. As of Bacula version 1.33 and the osst kernel driver version 0.9.14 or later, they now work. Please see the testing chapter as you must set a fixed block size.

QIC tapes are known to have a number of particularities (fixed block size, and one EOF rather than two to terminate the tape). As a consequence, you will need to take a lot of care in configuring them to make them work correctly with Bacula.

8.2 FreeBSD Users Be Aware!!!

Unless you have patched the pthreads library on FreeBSD 4.11 systems, you will lose data when Bacula spans tapes. This is because the unpatched pthreads library fails to return a warning status to Bacula that the end of the tape is near. This problem is fixed in FreeBSD systems released after 4.11. Please see the Tape Testing Chapter of this manual for **important** information on how to configure your tape drive for compatibility with Bacula.

8.3 Supported Autochangers

For information on supported autochangers, please see the Autochangers Known to Work with Bacula section of the Supported Autochangers chapter of this manual.

8.4 Tape Specifications

If you want to know what tape drive to buy that will work with Bacula, we really cannot tell you. However, we can say that if you are going to buy a drive, you should try to avoid DDS drives. The technology is rather old and DDS tape drives need frequent cleaning. DLT drives are generally much better (newer technology) and do not need frequent cleaning.

Below, you will find a table of DLT and LTO tape specifications that will give you some idea of the capacity and speed of modern tapes. The capacities that are listed are the native tape capacity without compression. All modern drives have hardware compression, and manufacturers often list compressed capacity using a compression ration of 2:1. The actual compression ratio will depend mostly on the data you have to backup, but I find that 1.5:1 is a much more reasonable number (i.e. multiply the value shown in the table by 1.5 to get a rough average of what you will probably see). The transfer rates are rounded to the nearest GB/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 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 **bw-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 **bw-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 [[slots] storage | dir | 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>]
*
```


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** or **dir** commands to walk up and down the directory tree and view what files will be restored. For example, if I enter **cd /home/kern/bacula/bacula-1.30** and then enter **dir** I will get a listing of all the files in the Bacula source directory. On your system, the path will be somewhat different. For more information on this, please refer to the Restore Command Chapter of this manual for more details.

To exit this mode, simply enter:

```
done
```

and you will get the following output:

```
Bootstrap records written to
/home/kern/bacula/testbin/working/restore.bsr
The restore job will require the following Volumes:

TestVolume001
1444 files selected to restore.
Run Restore job
JobName:      RestoreFiles
Bootstrap:    /home/kern/bacula/testbin/working/restore.bsr
Where:        /tmp/bacula-restores
Replace:      always
FileSet:      Full Set
Backup Client: rufus-fd
Restore Client: rufus-fd
Storage:      File
JobId:        *None*
When:         2005-04-28 14:53:54
OK to run? (yes/mod/no):
```

If you answer **yes** your files will be restored to **/tmp/bacula-restores**. If you want to restore the files to their original locations, you must use the **mod** option and explicitly set **Where:** to nothing (or to **/**). We recommend you go ahead and answer **yes** and after a brief moment, enter **messages**, at which point you should get a listing of all the files that were restored as well as a summary of the job that looks similar to this:

```
28-Apr-2005 14:56 rufus-dir: Bacula 2.1.8 (08May07): 08-May-2007 14:56:06
Build OS:          i686-pc-linux-gnu suse 10.2
JobId:             2
Job:               RestoreFiles.2007-05-08_14.56.06
Restore Client:    rufus-fd
Start time:        08-May-2007 14:56
End time:          08-May-2007 14:56
Files Restored:    1,444
Bytes Restored:    38,816,381
Rate:              9704.1 KB/s
FD Errors:         0
FD termination status: OK
SD termination status: OK
Termination:       Restore OK
08-May-2007 14:56 rufus-dir: Begin pruning Jobs.
08-May-2007 14:56 rufus-dir: No Jobs found to prune.
08-May-2007 14:56 rufus-dir: Begin pruning Files.
08-May-2007 14:56 rufus-dir: No Files found to prune.
08-May-2007 14:56 rufus-dir: End auto prune.
```

After exiting the Console program, you can examine the files in **/tmp/bacula-restores**, which will contain a small directory tree with all the files. Be sure to clean up at the end with:

```
rm -rf /tmp/bacula-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 {
    Name = rufus-fd
    FDport = 9102
    WorkingDirectory = /home/kern/bacula/working
    Pid Directory = /var/run
}
...
```

would become:

```
...
#
# "Global" File daemon configuration specifications
#
FileDaemon {
    Name = matou-fd
    FDport = 9102
    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 numbers are authorized by IANA, and should not be used by other programs, but apparently some HP printers do use these port numbers. A **netstat -a** on the Storage daemon's machine can determine who is using the 9103 port (used for FD to SD communications in Bacula).

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.
- g ;group;** Run the daemon under this group. This must be a group name, not a GID.
- s** Do not trap signals. This option is needed to run the daemon under the debugger.
- t** Read the configuration file and print any error messages, then immediately exit. Useful for syntax testing of new configuration files.
- u ;user;** Run the daemon as this user. This must be a user name, not a UID.
- v** Be more verbose or more complete in printing error and informational messages. Recommended.
- ?** Print the version and list of options.

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	Recyl
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

The Restore Command

10.1 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.

10.2 The Restore Command

Since Bacula maintains a catalog of your files and on which Volumes (disk or tape), they are stored, it can do most of the bookkeeping work, allowing you simply to specify what kind of restore you want (current, before a particular date), and what files to restore. Bacula will then do the rest.

This is accomplished using the **restore** command in the Console. First you select the kind of restore you want, then the JobIds are selected, the File records for those Jobs are placed in an internal Bacula directory tree, and the restore enters a file selection mode that allows you to interactively walk up and down the file

tree selecting individual files to be restored. This mode is somewhat similar to the standard Unix **restore** program's interactive file selection mode.

If a Job's file records have been pruned from the catalog, the **restore** command will be unable to find any files to restore. Bacula will ask if you want to restore all of them or if you want to use a regular expression to restore only a selection while reading media. See FileRegex option and below for more details on this.

Within the Console program, after entering the **restore** command, you are presented with the following selection prompt:

```
First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.
To select the JobIds, you have the following choices:
  1: List last 20 Jobs run
  2: List Jobs where a given File is saved
  3: Enter list of comma separated JobIds to select
  4: Enter SQL list command
  5: Select the most recent backup for a client
  6: Select backup for a client before a specified time
  7: Enter a list of files to restore
  8: Enter a list of files to restore before a specified time
  9: Find the JobIds of the most recent backup for a client
 10: Find the JobIds for a backup for a client before a specified time
 11: Enter a list of directories to restore for found JobIds
 12: Cancel
Select item: (1-12):
```

There are a lot of options, and as a point of reference, most people will want to select 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 to 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 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 | Lev1 | 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**.

10.2.1 Restore a pruned job using a pattern

During a restore, if all File records are pruned from the catalog for a Job, normally Bacula can restore only all files saved. That is there is no way using the catalog to select individual files. With this new feature, Bacula will ask if you want to specify a Regex expression for extracting only a part of the full backup.

```
Building directory tree for JobId(s) 1,3 ...
There were no files inserted into the tree, so file selection
is not possible. Most likely your retention policy pruned the files

Do you want to restore all the files? (yes|no): no

Regex matching files to restore? (empty to abort): /tmp/regress/(bin|tests)/
Bootstrap records written to /tmp/regress/working/zog4-dir.restore.1.bsr
```

See also FileRegex bsr option for more information.

10.3 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.

10.4 Replace Options

When restoring, you have the option to specify a Replace option. This directive determines the action to be taken when restoring a file or directory that already exists. This directive can be set by selecting the **mod** option. You will be given a list of parameters to choose from. Full details on this option can be found in the Job Resource section of the Director documentation.

10.5 Command Line Arguments

If all the above sounds complicated, you will probably agree that it really isn't after trying it a few times. It is possible to do everything that was shown above, with the exception of selecting the FileSet, by using command line arguments with a single command by entering:

```
restore client=Rufus select current all done yes
```

The **client=Rufus** specification will automatically select Rufus as the client, the **current** tells Bacula that you want to restore the system to the most current state possible, 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** – initially specifies the client from which the backup was made and the client to which the restore will be made. See also "restoreclient" keyword.
- **restoreclient=xxxx** – if the keyword is specified, then the restore is written to that client.
- **jobid=nnn** – specify a JobId or comma separated list of JobIds to be restored.
- **before=YYYY-MM-DD HH:MM:SS** – specify a date and time to which the system should be restored. Only Jobs started before the specified date/time will be selected, and as is the case for **current** Bacula will automatically find the most recent prior Full save and all Differential and Incremental saves run before the date you specify. Note, this command is not too user friendly in that you must specify the date/time exactly as shown.
- **file=filename** – specify a filename to be restored. You must specify the full path and filename. Prefixing the entry with a less-than sign (<) will cause Bacula to assume that the filename is on your system and contains a list of files to be restored. Bacula will thus read the list from that file. Multiple file=xxx specifications may be specified on the command line.
- **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.

10.6 Using File Relocation

10.6.1 Introduction

The **where=** option is simple, but not very powerful. With file relocation, Bacula can restore a file to the same directory, but with a different name, or in an other directory without recreating the full path.

You can also do filename and path manipulations, implemented in Bacula 2.1.8 or later, such as adding a suffix to all your files, renaming files or directories, etc. These options will overwrite **where=** option.

For example, many users use OS snapshot features so that file `/home/eric/mbox` will be backed up from the directory `/.snap/home/eric/mbox`, which can complicate restores. If you use **where=/tmp**, the file will be restored to `/tmp/.snap/home/eric/mbox` and you will have to move the file to `/home/eric/mbox.bkp` by hand.

However, case, you could use the **strip_prefix=/.snap** and **add_suffix=.bkp** options and Bacula will restore the file to its original location – that is `/home/eric/mbox`.

To use this feature, there are command line options as described in the restore section of this manual; you can modify your restore job before running it; or you can add options to your restore job in as described in `bacula-dir.conf`.

Parameters to modify:

- 1: Level
- 2: Storage
- ...
- 10: File Relocation
- ...

Select parameter to modify (1-12):

This will replace your current Where value

- 1: Strip prefix
- 2: Add prefix
- 3: Add file suffix
- 4: Enter a regexp
- 5: Test filename manipulation
- 6: Use this ?

Select parameter to modify (1-6):

10.6.2 RegexWhere Format

The format is very close to that used by sed or Perl (`s/replace this/by that/`) operator. A valid `regexwhere` expression has three fields :

- a search expression (with optionnal submatch)
- a replacement expression (with optionnal back references \$1 to \$9)
- a set of search options (only case-insensitive “i” at this time)

Each field is delimited by a separator specified by the user as the first character of the expression. The separator can be one of the following:

`<separator-keyword> = / ! ; % : , ~ # = &`

You can use several expressions separated by a commas.

Examples

Original filename	New filename	RegexWhere	Comments
c:/system.ini	c:/system.old.ini	/.ini\$/old.ini/	\$ matches end of name
/prod/u01/pdata/	/rect/u01/rdata	/prod/rect/,/pdata/rdata/	uses two regexp
/prod/u01/pdata/	/rect/u01/rdata	!/prod!/rect!/pdata/rdata/	use ! as separator
C:/WINNT	d:/WINNT	/c:/d:/i	case insensitive pattern match

10.7 Restoring Directory Attributes

Depending how you do the restore, you may or may not get the directory entries back to their original state. Here are a few of the problems you can encounter, and for some 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 than 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.

10.8 Restoring on Windows

If you are restoring on WinNT/2K/XP systems, Bacula will restore the files with the original ownerships and permissions as would be expected. This is also true if you are restoring those files to an alternate directory (using the Where option in restore). However, if the alternate directory does not already exist, the Bacula File daemon (Client) will try to create it. In some cases, it may not create the directories, and if it does since the File daemon runs under the SYSTEM account, the directory will be created with SYSTEM ownership and permissions. In this case, you may have problems accessing the newly restored files.

To avoid this problem, you should create any alternate directory before doing the restore. Bacula will not change the ownership and permissions of the directory if it is already created as long as it is not one of the directories being restored (i.e. written to tape).

The default restore location is `/tmp/bacula-restores/` and if you are restoring from drive **E:**, the default will be `/tmp/bacula-restores/e/`, so you should ensure that this directory exists before doing the restore, or use the **mod** option to select a different **where** directory that does exist.

Some users have experienced problems restoring files that participate in the Active Directory. They also report that changing the userid under which Bacula (bacula-fd.exe) runs, from SYSTEM to a Domain Admin userid, resolves the problem.

10.9 Restoring Files Can Be Slow

Restoring files is generally **much** slower than backing them up for several reasons. The first is that during a backup the tape is normally already positioned and Bacula only needs to write. On the other hand, because restoring files is done so rarely, Bacula keeps only the start file and block on the tape for the whole job rather than on a file by file basis which would use quite a lot of space in the catalog.

Bacula will forward space to the correct file mark on the tape for the Job, then forward space to the correct block, and finally sequentially read each record until it gets to the correct one(s) for the file or files you want to restore. Once the desired files are restored, Bacula will stop reading the tape.

Finally, instead of just reading a file for backup, during the restore, Bacula must create the file, and the operating system must allocate disk space for the file as Bacula is restoring it.

For all the above reasons the restore process is generally much slower than backing up (sometimes it takes three times as long).

10.10 Problems Restoring Files

The most frequent problems users have restoring files are error messages such as:

```
04-Jan 00:33 z217-sd: RestoreFiles.2005-01-04_00.31.04 Error:
block.c:868 Volume data error at 20:0! Short block of 512 bytes on
device /dev/tape discarded.
```

or

```
04-Jan 00:33 z217-sd: RestoreFiles.2005-01-04_00.31.04 Error:
block.c:264 Volume data error at 20:0! Wanted ID: "BB02", got ".".
Buffer discarded.
```

Both these kinds of messages indicate that you were probably running your tape drive in fixed block mode rather than variable block mode. Fixed block mode will work with any program that reads tapes sequentially such as tar, but Bacula repositions the tape on a block basis when restoring files because this will speed up the restore by orders of magnitude when only a few files are being restored. There are several ways that you can attempt to recover from this unfortunate situation.

Try the following things, each separately, and reset your Device resource to what it is now after each individual test:

1. Set "Block Positioning = no" in your Device resource and try the restore. This is a new directive and untested.
2. Set "Minimum Block Size = 512" and "Maximum Block Size = 512" and try the restore. If you are able to determine the block size your drive was previously using, you should try that size if 512 does not work. This is a really horrible solution, and it is not at all recommended to continue backing up your data without correcting this condition. Please see the Tape Testing chapter for more on this.

3. Try editing the `restore.bsr` file at the `Run xxx yes/mod/no` prompt before starting the restore job and remove all the `VolBlock` statements. These are what causes Bacula to reposition the tape, and where problems occur if you have a fixed block size set for your drive. The `VolFile` commands also cause repositioning, but this will work regardless of the block size.
4. Use `bextract` to extract the files you want – it reads the Volume sequentially if you use the include list feature, or if you use a `.bsr` file, but remove all the `VolBlock` statements after the `.bsr` file is created (at the `Run yes/mod/no`) prompt but before you start the restore.

10.11 Restore Errors

There are a number of reasons why there may be restore errors or warning messages. Some of the more common ones are:

file count mismatch This can occur for the following reasons:

- You requested Bacula not to overwrite existing or newer files.
- A Bacula miscount of files/directories. This is an on-going problem due to the complications of directories, soft/hard link, and such. Simply check that all the files you wanted were actually restored.

file size error When Bacula restores files, it checks that the size of the restored file is the same as the file status data it saved when starting the backup of the file. If the sizes do not agree, Bacula will print an error message. This size mismatch most often occurs because the file was being written as Bacula backed up the file. In this case, the size that Bacula restored will be greater than the status size. This often happens with log files.

If the restored size is smaller, then you should be concerned about a possible tape error and check the Bacula output as well as your system logs.

10.12 Example Restore Job Resource

```
Job {
  Name = "RestoreFiles"
  Type = Restore
  Client = Any-client
  FileSet = "Any-FileSet"
  Storage = Any-storage
  Where = /tmp/bacula-restores
  Messages = Standard
  Pool = Default
}
```

If **Where** is not specified, the default location for restoring files will be their original locations.

10.13 File Selection Commands

After you have selected the Jobs to be restored and Bacula has created the in-memory directory tree, you will enter file selection mode as indicated by the dollar sign (\$) prompt. While in this mode, you may use the commands listed above. The basic idea is to move up and down the in memory directory structure with the `cd` command much as you normally do on the system. Once you are in a directory, you may select the files that you want restored. As a default no files are marked to be restored. If you wish to start with all files, simply enter: `cd /` and `mark *`. Otherwise proceed to select the files you wish to restore by marking them with the `mark` command. The available commands are:

cd The **cd** command changes the current directory to the argument specified. It operates much like the Unix **cd** command. Wildcard specifications are not permitted.

Note, on Windows systems, the various drives (c:, d:, ...) are treated like a directory within the file tree while in the file selection mode. As a consequence, you must do a **cd c:** or possibly in some cases a **cd C:** (note upper case) to get down to the first directory.

dir The **dir** command is similar to the **ls** command, except that it prints it in long format (all details). This command can be a bit slower than the **ls** command because it must access the catalog database for the detailed information for each file.

estimate The **estimate** command prints a summary of the total files in the tree, how many are marked to be restored, and an estimate of the number of bytes to be restored. This can be useful if you are short on disk space on the machine where the files will be restored.

find The **find** command accepts one or more arguments and displays all files in the tree that match that argument. The argument may have wildcards. It is somewhat similar to the Unix command **find / -name arg**.

ls The **ls** command produces a listing of all the files contained in the current directory much like the Unix **ls** command. You may specify an argument containing wildcards, in which case only those files will be listed.

Any file that is marked to be restored will have its name preceded by an asterisk (*). Directory names will be terminated with a forward slash (/) to distinguish them from filenames.

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.

10.14 Restoring When Things Go Wrong

This and the following sections will try to present a few of the kinds of problems that can come up making restoring more difficult. We will try to provide a few ideas how to get out of these problem situations. In addition to what is presented here, there is more specific information on restoring a Client and your Server in the Disaster Recovery Using Bacula chapter of this manual.

Problem My database is broken.

Solution For 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 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:

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 "l1ist jobid=nn" command, you can obtain all the important information about the job:

```
l1ist 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 11

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 the Volume Status is **Purged**. At that point, it will choose the oldest purged volume and recycle it.

If there are no volumes with Status **Purged**, then the recycling occurs in two steps: The first is that the Catalog for a Volume must be pruned of all Jobs (i.e. Purged). Files contained on that Volume, and the second step is the actual recycling of the Volume. Only Volumes marked **Full** or **Used** will be considered for pruning. The Volume will be purged 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.

11.1 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).

11.2 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 autopruning, 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 as the Volume remains appendable. The Volume Retention period count down begins only when the Append status has been changed to some other 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.

11.3 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 directive 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".

11.4 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.

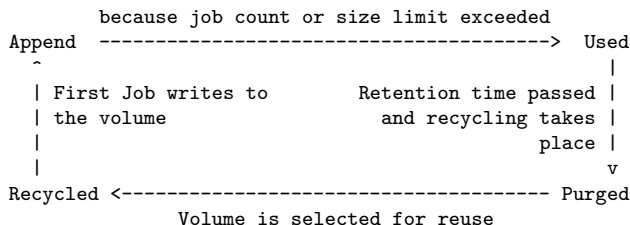
VolumeNa	Media	VolSta	VolByte	LastWritten	VolRet	Rec
File0001	File	Full	4190055	2002-05-25	14400	0

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:



11.5 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
}

```

11.6 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 pre-labeled 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 {
    ffffffffffffffffff
  }
  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
}

```

11.7 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 = XXXXXXXXXXXX
    SDPort = 9103
    Password = XXXXXXXXXXXX
    Device = FileStorage
    Media Type = File
}
FileSet {
    Name = "Test Files"
    Include = signature=MD5 {
        ffffffffffffffff
    }
    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 **ffffff**'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.

11.8 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 12

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.

12.1 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.

12.1.1 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.

12.1.2 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.

12.1.3 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.

12.2 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.

12.3 An Example

The following example is not very practical, but can be used to demonstrate the proof of concept in a relatively short period of time. The example consists of a two clients that are backed up to a set of 12 archive files (Volumes) for each client into different directories on the Storage machine. Each Volume is used (written) only once, and there are four Full saves done every hour (so the whole thing cycles around after three hours).

What is key here is that each physical device on the Storage daemon has a different Media Type. This allows the Director to choose the correct device for restores ...

The Director's configuration file is as follows:

```
Director {
  Name = my-dir
  QueryFile = "/bacula/bin/query.sql"
  PidDirectory = "~/bacula/working"
  WorkingDirectory = "~/bacula/working"
  Password = dir_password
}
Schedule {
  Name = "FourPerHour"
  Run = Level=Full hourly at 0:05
  Run = Level=Full hourly at 0:20
  Run = Level=Full hourly at 0:35
  Run = Level=Full hourly at 0:50
}
Job {
  Name = "RecycleExample"
  Type = Backup
  Level = Full
  Client = Rufus
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = FileStorage
  Pool = Recycle
  Schedule = FourPerHour
}

Job {
  Name = "RecycleExample2"
  Type = Backup
  Level = Full
  Client = Roxie
  FileSet= "Example FileSet"
  Messages = Standard
  Storage = FileStorage1
  Pool = Recycle1
  Schedule = FourPerHour
}

FileSet {
  Name = "Example FileSet"
  Include {
    Options {
      compression=GZIP
      signature=SHA1
    }
  }
}
```



```

        File = /home/kern/bacula/bin
    }
}

Client {
    Name = Rufus
    Address = rufus
    Catalog = BackupDB
    Password = client_password
}

Client {
    Name = Roxie
    Address = roxie
    Catalog = BackupDB
    Password = client1_password
}

Storage {
    Name = FileStorage
    Address = rufus
    Password = local_storage_password
    Device = RecycleDir
    Media Type = File
}

Storage {
    Name = FileStorage1
    Address = rufus
    Password = local_storage_password
    Device = RecycleDir1
    Media Type = File1
}

Catalog {
    Name = BackupDB
    dbname = bacula; user = bacula; password = ""
}

Messages {
    Name = Standard
    ...
}

Pool {
    Name = Recycle
    Use Volume Once = yes
    Pool Type = Backup
    LabelFormat = "Recycle-"
    AutoPrune = yes
    VolumeRetention = 2h
    Maximum Volumes = 12
    Recycle = yes
}

Pool {
    Name = Recycle1
    Use Volume Once = yes
    Pool Type = Backup
    LabelFormat = "Recycle1-"
    AutoPrune = yes
    VolumeRetention = 2h
    Maximum Volumes = 12
    Recycle = yes
}

```

and the Storage daemon's configuration file is:

```

Storage {
    Name = my-sd
    WorkingDirectory = "~/bacula/working"
    Pid Directory = "~/bacula/working"
    MaximumConcurrentJobs = 10
}

Director {

```

```

    Name = my-dir
    Password = local_storage_password
}
Device {
    Name = RecycleDir
    Media Type = File
    Archive Device = /home/bacula/backups
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}

Device {
    Name = RecycleDir1
    Media Type = File1
    Archive Device = /home/bacula/backups1
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}

Messages {
    Name = Standard
    director = my-dir = all
}

```

With a little bit of work, you can change the above example into a weekly or monthly cycle (take care about the amount of archive disk space used).

12.4 Backing up to Multiple Disks

Bacula can, of course, use multiple disks, but in general, each disk must be a separate Device specification in the Storage daemon's conf file, and you must then select what clients to backup to each disk. You will also want to give each Device specification a different Media Type so that during a restore, Bacula will be able to find the appropriate drive.

The situation is a bit more complicated if you want to treat two different physical disk drives (or partitions) logically as a single drive, which Bacula does not directly support. However, it is possible to back up your data to multiple disks as if they were a single drive by linking the Volumes from the first disk to the second disk.

For example, assume that you have two disks named **/disk1** and **/disk2**. If you then create a standard Storage daemon Device resource for backing up to the first disk, it will look like the following:

```

Device {
    Name = client1
    Media Type = File
    Archive Device = /disk1
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}

```

Since there is no way to get the above Device resource to reference both **/disk1** and **/disk2** we do it by pre-creating Volumes on **/disk2** with the following:

```

ln -s /disk2/Disk2-vol001 /disk1/Disk2-vol001
ln -s /disk2/Disk2-vol002 /disk1/Disk2-vol002

```

```
ln -s /disk2/Disk2-vol003 /disk1/Disk2-vol003
...
```

At this point, you can label the Volumes as Volume **Disk2-vol001**, **Disk2-vol002**, ... and Bacula will use them as if they were on /disk1 but actually write the data to /disk2. The only minor inconvenience with this method is that you must explicitly name the disks and cannot use automatic labeling unless you arrange to have the labels exactly match the links you have created.

An important thing to know is that Bacula treats disks like tape drives as much as it can. This means that you can only have a single Volume mounted at one time on a disk as defined in your Device resource in the Storage daemon's conf file. You can have multiple concurrent jobs running that all write to the one Volume that is being used, but if you want to have multiple concurrent jobs that are writing to separate disks drives (or partitions), you will need to define separate Device resources for each one, exactly as you would do for two different tape drives. There is one fundamental difference, however. The Volumes that you create on the two drives cannot be easily exchanged as they can for a tape drive, because they are physically resident (already mounted in a sense) on the particular drive. As a consequence, you will probably want to give them different Media Types so that Bacula can distinguish what Device resource to use during a restore. An example would be the following:

```
Device {
    Name = Disk1
    Media Type = File1
    Archive Device = /disk1
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}

Device {
    Name = Disk2
    Media Type = File2
    Archive Device = /disk2
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}
```

With the above device definitions, you can run two concurrent jobs each writing at the same time, one to /disk1 and the other to /disk2. The fact that you have given them different Media Types will allow Bacula to quickly choose the correct Storage resource in the Director when doing a restore.

12.5 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 Media Type (e.g. **File1**).

In this example, we have two clients, each with a different Pool and a different number of archive files retained. They also write to different directories with different Volume labeling.

The Director's configuration file is as follows:

```
Director {
    Name = my-dir
    QueryFile = "~/bacula/bin/query.sql"
    PidDirectory = "~/bacula/working"
    WorkingDirectory = "~/bacula/working"
    Password = dir_password
}
# Basic weekly schedule
Schedule {
    Name = "WeeklySchedule"
    Run = Level=Full fri at 1:30
    Run = Level=Incremental sat-thu at 1:30
}
FileSet {
    Name = "Example FileSet"
    Include {
Options {
compression=GZIP
signature=SHA1
}
        File = /home/kern/bacula/bin
    }
}
Job {
    Name = "Backup-client1"
    Type = Backup
    Level = Full
    Client = client1
    FileSet= "Example FileSet"
    Messages = Standard
    Storage = File1
    Pool = client1
    Schedule = "WeeklySchedule"
}
Job {
    Name = "Backup-client2"
    Type = Backup
    Level = Full
    Client = client2
    FileSet= "Example FileSet"
    Messages = Standard
    Storage = File2
    Pool = client2
    Schedule = "WeeklySchedule"
}
Client {
    Name = client1
    Address = client1
    Catalog = BackupDB
    Password = client1_password
    File Retention = 7d
}
Client {
    Name = client2
    Address = client2
    Catalog = BackupDB
    Password = client2_password
}
# Two Storage definitions with different Media Types
# permits different directories
Storage {
    Name = File1
    Address = rufus
    Password = local_storage_password
    Device = client1
    Media Type = File1
}
Storage {
    Name = File2
```

```

    Address = rufus
    Password = local_storage_password
    Device = client2
    Media Type = File2
}
Catalog {
    Name = BackupDB
    dbname = bacula; user = bacula; password = ""
}
Messages {
    Name = Standard
    ...
}
# Two pools permits different cycling periods and Volume names
# Cycle through 15 Volumes (two weeks)
Pool {
    Name = client1
    Use Volume Once = yes
    Pool Type = Backup
    LabelFormat = "Client1-"
    AutoPrune = yes
    VolumeRetention = 13d
    Maximum Volumes = 15
    Recycle = yes
}
# Cycle through 8 Volumes (1 week)
Pool {
    Name = client2
    Use Volume Once = yes
    Pool Type = Backup
    LabelFormat = "Client2-"
    AutoPrune = yes
    VolumeRetention = 6d
    Maximum Volumes = 8
    Recycle = yes
}

```

and the Storage daemon's configuration file is:

```

Storage {
    Name = my-sd
    WorkingDirectory = "~/bacula/working"
    Pid Directory = "~/bacula/working"
    MaximumConcurrentJobs = 10
}
Director {
    Name = my-dir
    Password = local_storage_password
}
# Archive directory for Client1
Device {
    Name = client1
    Media Type = File1
    Archive Device = /home/bacula/client1
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}
# Archive directory for Client2
Device {
    Name = client2
    Media Type = File2
    Archive Device = /home/bacula/client2
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}
Messages {
    Name = Standard
    director = my-dir = all
}

```


Chapter 13

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.

13.1 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.

13.2 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 ($15\text{GB} \times 6 + 2\text{GB} \times 5 + 0.3 \times 7 = 102.1\text{GB}$).

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.

13.3 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.

13.3.1 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.

13.3.2 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 concerning the Full pool for how to handle multiple clients.

13.3.3 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.

13.4 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 14

Migration and Copy

The term Migration, as used in the context of Bacula, means moving data from one Volume to another. In particular it refers to a Job (similar to a backup job) that reads data that was previously backed up to a Volume and writes it to another Volume. As part of this process, the File catalog records associated with the first backup job are purged. In other words, Migration moves Bacula Job data from one Volume to another by reading the Job data from the Volume it is stored on, writing it to a different Volume in a different Pool, and then purging the database records for the first Job.

The Copy process is essentially identical to the Migration feature with the exception that the Job that is copied is left unchanged. This essentially creates two identical copies of the same backup. However, the copy is treated as a copy rather than a backup job, and hence is not directly available for restore. If bacula finds a copy when a job record is purged (deleted) from the catalog, it will promote the copy as *real* backup and will make it available for automatic restore.

The Copy and the Migration jobs run without using the File daemon by copying the data from the old backup Volume to a different Volume in a different Pool.

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).

14.1 Migration and Copy Job Resource Directives

The following directives can appear in a Director's Job resource, and they are used to define a Migration job.

Pool = <Pool-name> The Pool specified in the Migration control Job is not a new directive for the Job resource, but it is particularly important because it determines what Pool will be examined for finding JobIds to migrate. The exception to this is when **Selection Type = SQLQuery**, 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.

Type = Copy **Copy** is a new type that defines the job that is run as being a Copy Job. A Copy Job is a sort of control job and does not have any Files associated with it, and in that sense they are more or less like an Admin job. Copy jobs simply check to see if there is anything to Copy then possibly start and control new Backup jobs to copy the data from the specified Pool to another Pool.

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.

PoolUncopiedJobs This selection which copies all jobs from a pool to an other pool which were not copied before is available only for copy Jobs.

Selection Pattern = <Quoted-string> The Selection Patterns permitted for each Selection-type-keyword are described above.

For the OldestVolume and SmallestVolume, this Selection pattern is not used (ignored).

For the Client, Volume, and Job keywords, this pattern must be a valid regular expression that will filter the appropriate item names found in the Pool.

For the SQLQuery keyword, this pattern must be a valid SELECT SQL statement that returns JobIds.

14.2 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, ...).

14.3 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.

14.4 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 = "ccV3lVTsQRsdIUGyab0N4sMDavui2h0BkmpBU0aQK0r9"
    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 = "ccV3lVTsQRsdIUGyab0N4sMDavui2h0BkmpBU0aQK0r9"
    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 DLTDive because the **Next Pool** in the Default Pool specifies that Migrations should go to the pool named **Tape**, which uses Storage **DLTDive**.

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 15

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.

15.1 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.

15.1.1 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.

15.1.2 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.

15.1.3 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.

15.2 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 <&lt;END_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
```

15.3 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.

15.3.1 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.

15.3.2 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.

15.3.3 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 16

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 be 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.

16.1 Knowing What SCSI Devices You Have

Under Linux, you can

```
cat /proc/scsi/scsi
```

to see what SCSI devices you have available. You can also:

```
cat /proc/scsi/sg/device_hdr /proc/scsi/sg/devices
```

to find out how to specify their control address (**/dev/sg0** for the first, **/dev/sg1** for the second, ...) on the **Changer Device =** Bacula directive.

You can also use the excellent **lsscsi** tool.

```
$ lsscsi -g
[1:0:2:0]   tape    SEAGATE  ULTRIUM06242-XXX 1619 /dev/st0 /dev/sg9
[1:0:14:0]  mediumx STK      L180              0315 /dev/sch0 /dev/sg10
[2:0:3:0]   tape    HP       Ultrium 3-SCSI   G24S /dev/st1 /dev/sg11
[3:0:0:0]   enclosu HP      A6255A           HP04  -      /dev/sg3
[3:0:1:0]   disk    HP      36.4G ST336753FC    HP00  /dev/sdd /dev/sg4
```

For more detailed information on what SCSI devices you have please see the Linux SCSI Tricks section 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.

16.2 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.

16.3 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.

16.4 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.

16.5 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 17

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.

17.1 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.

17.2 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.

17.3 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.

17.4 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.

17.5 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.

17.6 Simulating Barcodes in your Autochanger

You can simulate barcodes in your autochanger by making the **mtx-changer** script return the same information that an autochanger with barcodes would do. This is done by commenting out the one and only line in the **list**) case, which is:

```
{MTX} -f $ctl status | grep " *Storage Element [0-9]*:.*Full" | awk "{print \$3 \$4}" | sed "s/Full *\([:VolumeTag=)\)*//"
```

at approximately line 99 by putting a **#** in column one of that line, or by simply deleting it. Then in its place add a new line that prints the contents of a file. For example:

```
cat /etc/bacula/changer.volumes
```

Be sure to include a full path to the file, which can have any name. The contents of the file must be of the following format:

```
1:Volume1
2:Volume2
3:Volume3
...
```

Where the 1, 2, 3 are the slot numbers and Volume1, Volume2, ... are the Volume names in those slots. You can have multiple files that represent the Volumes in different magazines, and when you change magazines, simply copy the contents of the correct file into your **/etc/bacula/changer.volumes** file. There is no need to stop and start Bacula when you change magazines, simply put the correct data in the file, then run the **update slots** command, and your autochanger will appear to Bacula to be an autochanger with barcodes.

17.7 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.

17.8 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.

17.9 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.

17.10 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
...								

17.11 Barcode Support

Bacula provides barcode support with two Console commands, **label barcodes** and **update slots**.

The **label barcodes** will cause Bacula to read the barcodes of all the cassettes that are currently installed in the magazine (cassette holder) using the **mtx-changer list** command. Each cassette is mounted in turn and labeled with the same Volume name as the barcode.

The **update slots** command will first obtain the list of cassettes and their barcodes from **mtx-changer**. Then it will find each volume in turn in the catalog database corresponding to the barcodes and set its Slot to correspond to the value just read. If the Volume is not in the catalog, then nothing will be done. This command is useful for synchronizing Bacula with the current magazine in case you have changed magazines

or in case you have moved cassettes from one slot to another. If the autochanger is empty, nothing will be done.

The **Cleaning Prefix** statement can be used in the Pool resource to define a Volume name prefix, which if it matches that of the Volume (barcode) will cause that Volume to be marked with a VolStatus of **Cleaning**. This will prevent Bacula from attempting to write on the Volume.

17.12 Use bconsole to display Autochanger content

The **status slots storage=xxx** command displays autochanger content.

Slot	Volume Name	Status	Type	Pool	Loaded
1	00001	Append	DiskChangerMedia	Default	0
2	00002	Append	DiskChangerMedia	Default	0
3*	00003	Append	DiskChangerMedia	Scratch	0
4					0

If you see a * near the slot number, you have to run **update slots** command to synchronize autochanger content with your catalog.

17.13 Bacula Autochanger Interface

Bacula calls the autochanger script that you specify on the **Changer Command** statement. Normally this script will be the **mtx-changer** script that we provide, but it can in fact be any program. The only requirement for the script is that it must understand the commands that Bacula uses, which are **loaded**, **load**, **unload**, **list**, and **slots**. In addition, each of those commands must return the information in the precise format as specified below:

```
- Currently the changer commands used are:
  loaded -- returns number of the slot that is loaded, base 1,
           in the drive or 0 if the drive is empty.
  load   -- loads a specified slot (note, some autochangers
           require a 30 second pause after this command) into
           the drive.
  unload -- unloads the device (returns cassette to its slot).
  list   -- returns one line for each cassette in the autochanger
           in the format <slot>:<barcode>. Where
           the {\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 18

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 Fast-Stor 2, Sun Storedge L8	8	200GB
Linux	BDT	AIT	BDT Thin-Stor	?	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 Packet-Loader 1x10 2U	10	80/160GB
-	Exabyte	LTO	Magnum 1x7 LTO Tape Autoloader	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 - re-branded 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 PowerLoader LTO-2	10-19	200/400GB
FreeBSD 5.4-Stable	Overland	LTO-2	Overland Powerloader tape	17	100GB
-	Overland	LTO	Overland Neo2000 LTO	26-30	100GB
Linux	Quantum	DLT-S4	Superloader 3	16	800/1600GB
Linux	Quantum	LTO-2	Superloader 3	16	200/400GB
Linux	Quantum	LTO-3	PX502	??	??
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 Desktop Archive Python 29279	4	20GB
Linux	Tandberg	DLT VI	VS 640	8?	35/70GB

Linux 2.6.x	Tandberg Data	SLR100	SLR100 Au- toloader	8	50/100GB
----------------	------------------	--------	------------------------	---	----------

Chapter 19

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 despoiled 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.

19.1 Data Spooling Directives

The following directives can be used to control data spooling.

- To turn data spooling on/off at the Job level in the Job resource in the Director's conf file (default **no**).
SpoolData = yes—no
- To override the Job specification in a Schedule Run directive in the Director's conf file.
SpoolData = yes—no

- To override the Job specification in a bconsole session using the **run** command. Please note that this does **not** refer to a configuration statement, but to an argument for the run command.

SpoolData=yes—no

- To limit the the maximum spool file size for a particular job in the Job resource
Spool Size = size Where size is a the maximum spool size for this job specified in bytes.
- To limit the maximum total size of the spooled data for a particular device. Specified in the Device resource of the Storage daemon's conf file (default unlimited).

Maximum Spool Size = size Where size is a the maximum spool size for all jobs specified in bytes.

- To limit the maximum total size of the spooled data for a particular device for a single job. Specified in the Device Resource of the Storage daemon's 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

19.2 !!! 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.

19.3 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 20

Using Bacula catalog to grab information

Bacula catalog contains lot of information about your IT infrastructure, how many files, their size, the number of video or music files etc. Using Bacula catalog during the day to get them permit to save resources on your servers.

In this chapter, you will find tips and information to measure bacula efficiency and report statistics.

20.1 Job statistics

If you (or probably your boss) want to have statistics on your backups to provide some *Service Level Agreement* indicators, you could use a few SQL queries on the Job table to report how many:

- jobs have run
- jobs have been successful
- files have been backed up
- ...

However, these statistics are accurate only if your job retention is greater than your statistics period. Ie, if jobs are purged from the catalog, you won't be able to use them.

Now, you can use the **update stats** [**days=num**] console command to fill the JobHistory table with new Job records. If you want to be sure to take in account only **good jobs**, ie if one of your important job has failed but you have fixed the problem and restarted it on time, you probably want to delete the first *bad* job record and keep only the successful one. For that simply let your staff do the job, and update JobHistory table after two or three days depending on your organization using the [**days=num**] option.

These statistics records aren't used for restoring, but mainly for capacity planning, billings, etc.

The Bweb interface provides a statistics module that can use this feature. You can also use tools like Talend or extract information by yourself.

The **Statistics Retention = <time>** director directive defines the length of time that Bacula will keep statistics job records in the Catalog database after the Job End time. (In JobHistory table) When this time period expires, and if user runs **prune stats** command, Bacula will prune (remove) Job records that are older than the specified period.

You can use the following Job resource in your nightly **BackupCatalog** job to maintain statistics.


```
Job {  
  Name = BackupCatalog  
  ...  
  RunScript {  
    Console = "update stats days=3"  
    Console = "prune stats yes"  
    RunsWhen = After  
    RunsOnClient = no  
  }  
}
```

Chapter 21

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.

21.1 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**.

21.2 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.

21.3 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 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.

21.4 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.

21.5 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.

21.6 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 22

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.

22.1 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.

22.2 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 23

The Windows Version of Bacula

At the current time only the File daemon or Client program has been thoroughly 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.

23.1 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 bwxc-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 bwxc-console or bconsole.

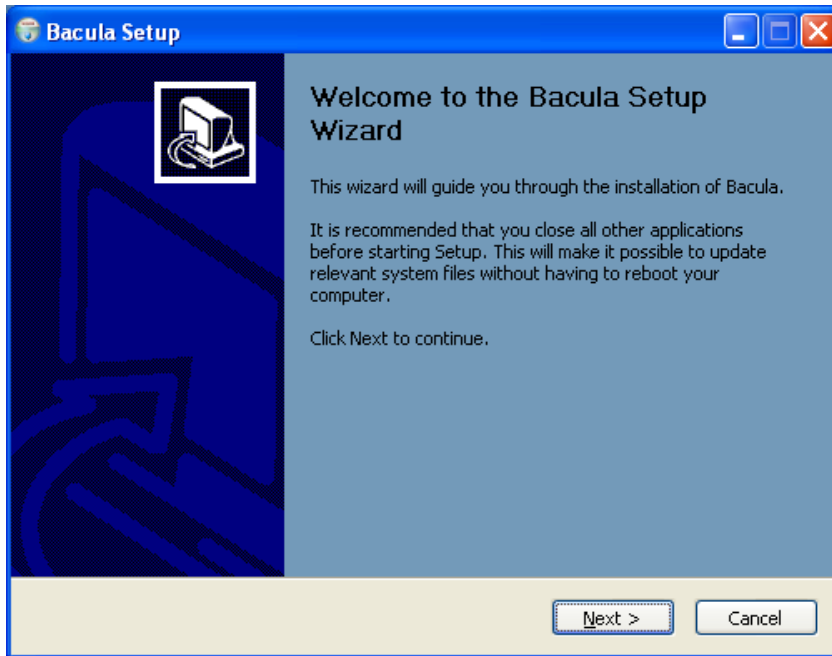
Finally, proceed with the installation.

- 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.

23.2 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.

23.3 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.

23.4 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.

23.5 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). 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.

23.6 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 - 39.pdd
  SDRReadSeqNo=5 fd=352
```

Here, you see under Running Jobs that JobId 1 is "VSS Backup Job started ..." This means that VSS is enabled for that job. If VSS is not enabled, it will simply show "Backup Job started ..." without the letters VSS.

The second way to know that the job was backed up with VSS is to look at the Job Report, which will look something like the following:

```
23-Jul 13:25 rufus-dir: Start Backup JobId 1, Job=NightlySave.2005-07-23_13.25.45
23-Jul 13:26 rufus-sd: Wrote label to prelabeled Volume "TestVolume001" on device "DDS-4" (/dev/nst0)
23-Jul 13:26 rufus-sd: Spooling data ...
23-Jul 13:26 Tibs: Generate VSS snapshots. Driver="VSS WinXP", Drive(s)="C"
23-Jul 13:26 Tibs: VSS Writer: "MSDEWriter", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "Microsoft Writer (Bootable State)", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "WMI Writer", State: 1 (VSS_WS_STABLE)
23-Jul 13:26 Tibs: VSS Writer: "Microsoft Writer (Service State)", State: 1 (VSS_WS_STABLE)
```

In the above Job Report listing, you see that the VSS snapshot was generated for drive C (if other drives are backed up, they will be listed on the **Drive(s)="C"**) You also see the reports from each of the writer program. Here they all report VSS_WS_STABLE, which means that you will get a consistent snapshot of the data handled by that writer.

23.7 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).

23.8 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.

23.9 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.

23.10 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/>.

23.11 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.

23.12 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.

23.13 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:

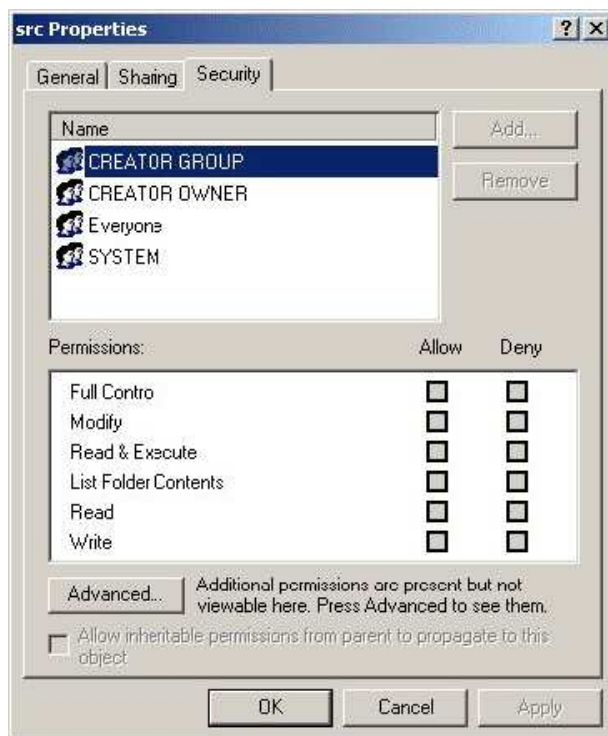


If you encounter this message, the following steps will change the permissions to allow full access.

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.

23.14 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.

23.15 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.

23.16 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.

23.17 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 24

Disaster Recovery Using Bacula

24.1 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.

24.2 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?

24.3 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.

24.4 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)

24.5 Requirements

24.6 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 ...

24.7 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.

24.8 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.

24.9 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.

24.10 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) preferably 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**.

24.11 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 be 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`

24.12 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.

24.13 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`

24.14 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.

24.15 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.

24.16 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/>.

24.17 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.
```

24.18 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
```

24.19 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 25

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`

25.1 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
```

25.2 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://tinycs.sm-zone.net/>.

25.3 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.

25.4 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 26

Data Encryption

Bacula permits file data encryption and signing within the File Daemon (or Client) prior to sending data to the Storage Daemon. Upon restoration, file signatures are validated and any mismatches are reported. At no time does the Director or the Storage Daemon have access to unencrypted file contents.

It is very important to specify what this implementation does NOT do:

- There is one important restore problem to be aware of, namely, it's possible for the director to restore new keys or a Bacula configuration file to the client, and thus force later backups to be made with a compromised key and/or with no encryption at all. You can avoid this by not changing the location of the keys in your Bacula File daemon configuration file, and not changing your File daemon keys. If you do change either one, you must ensure that no restore is done that restores the old configuration or the old keys. In general, the worst effect of this will be that you can no longer connect the File daemon.
- The implementation does not encrypt file metadata such as file path names, permissions, and ownership. Extended attributes are also currently not encrypted. However, Mac 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 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.

26.1 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 ...
```

26.2 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.

26.3 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`
- 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.

26.4 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.

26.5 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 27

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.

27.1 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.

27.2 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: DLDrive
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.

27.3 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.

27.4 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 28

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.

28.1 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 least one Volume specification is required.

Count The value is the total number of files that will be restored for this Volume. This allows the Storage daemon to know when to stop reading the Volume. This value is optional.

VolFile The value is a file number, a list of file numbers, or a range of file numbers to match on the current Volume. The file number represents the physical file on the Volume where the data is stored. For a tape volume, this record is used to position to the correct starting file, and once the tape is past the last specified file, reading will stop.

VolBlock The value is a block number, a list of block numbers, or a range of block numbers to match on the current Volume. The block number represents the physical block within the file on the Volume where the data is stored.

VolSessionTime The value specifies a Volume Session Time to be matched from the current volume.

VolSessionId The value specifies a VolSessionId, a list of volume session ids, or a range of volume session ids to be matched from the current Volume. Each VolSessionId and VolSessionTime pair corresponds to a unique Job that is backed up on the Volume.

JobId The value specifies a JobId, list of JobIds, or range of JobIds to be selected from the current Volume. Note, the JobId may not be unique if you have multiple Directors, or if you have reinitialized your database. The JobId filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

Job The value specifies a Job name or list of Job names to be matched on the current Volume. The Job corresponds to a unique VolSessionId and VolSessionTime pair. However, the Job is perhaps a bit more readable by humans. Standard regular expressions (wildcards) may be used to match Job names. The Job filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

Client The value specifies a Client name or list of Clients to will be matched on the current Volume. Standard regular expressions (wildcards) may be used to match Client names. The Client filter works only if you do not run multiple simultaneous jobs. This value is optional and not used by Bacula to restore files.

FileIndex The value specifies a FileIndex, list of FileIndexes, or range of FileIndexes to be selected from the current Volume. Each file (data) stored on a Volume within a Session has a unique FileIndex. For each Session, the first file written is assigned FileIndex equal to one and incremented for each file backed up.

This for a given Volume, the triple VolSessionId, VolSessionTime, and FileIndex uniquely identifies a file stored on the Volume. Multiple copies of the same file may be stored on the same Volume, but for each file, the triple VolSessionId, VolSessionTime, and FileIndex will be unique. This triple is stored in the Catalog database for each file.

To restore a particular file, this value (or a range of FileIndexes) is required.

FileRegex The value is a regular expression. When specified, only matching filenames will be restored.

```
FileRegex=~etc/passwd(.old)?
```

Slot The value specifies the autochanger slot. There may be only a single **Slot** specification for each Volume.

Stream The value specifies a Stream, a list of Streams, or a range of Streams to be selected from the current Volume. Unless you really know what you are doing (the internals of **Bacula**), you should avoid this specification. This value is optional and not used by Bacula to restore files.

***JobType** Not yet implemented.

***JobLevel** Not yet implemented.

The **Volume** record is a bit special in that it must be the first record. The other keyword records may appear in any order and any number following a Volume record.

Multiple Volume records may be specified in the same bootstrap file, but each one starts a new set of filter criteria for the Volume.

In processing the bootstrap file within the current Volume, each filter specified by a keyword is **AND**ed with the next. Thus,

```
Volume = Test-01
Client = "My machine"
FileIndex = 1
```

will match records on Volume **Test-01** **AND** Client records for **My machine** **AND** FileIndex equal to **one**.

Multiple occurrences of the same record are **OR**ed together. Thus,

```
Volume = Test-01
Client = "My machine"
Client = "Backup machine"
FileIndex = 1
```

will match records on Volume **Test-01** **AND** (Client records for **My machine** **OR** **Backup machine**) **AND** FileIndex equal to **one**.

For integer values, you may supply a range or a list, and for all other values except Volumes, you may specify a list. A list is equivalent to multiple records of the same keyword. For example,

```
Volume = Test-01
Client = "My machine", "Backup machine"
FileIndex = 1-20, 35
```

will match records on Volume **Test-01** **AND** (Client records for **My machine** **OR** **Backup machine**) **AND** (FileIndex 1 **OR** 2 **OR** 3 ... **OR** 20 **OR** 35).

As previously mentioned above, there may be multiple Volume records in the same bootstrap file. Each new Volume definition begins a new set of filter conditions that apply to that Volume and will be **OR**ed with any other Volume definitions.

As an example, suppose we query for the current set of tapes to restore all files on Client **Rufus** using the **query** command in the console program:

```
Using default Catalog name=MySQL DB=bacula
*query
Available queries:
  1: List Job totals:
  2: List where a file is saved:
  3: List where the most recent copies of a file are saved:
  4: List total files/bytes by Job:
  5: List total files/bytes by Volume:
  6: List last 10 Full Backups for a Client:
  7: List Volumes used by selected JobId:
  8: List Volumes to Restore All Files:
Choose a query (1-8): 8
Enter Client Name: Rufus

+-----+-----+-----+-----+-----+-----+
| JobId | StartTime          | VolumeName | StartFile | VolSesId | VolSesTime |
+-----+-----+-----+-----+-----+-----+
| 154   | 2002-05-30 12:08   | test-02    | 0          | 1         | 1022753312 |
| 202   | 2002-06-15 10:16   | test-02    | 0          | 2         | 1024128917 |
| 203   | 2002-06-15 11:12   | test-02    | 3          | 1         | 1024132350 |
| 204   | 2002-06-18 08:11   | test-02    | 4          | 1         | 1024380678 |
+-----+-----+-----+-----+-----+-----+
```

The output shows us that there are four Jobs that must be restored. The first one is a Full backup, and the following three are all Incremental backups.

The following bootstrap file will restore those files:

```

Volume=test-02
VolSessionId=1
VolSessionTime=1022753312
Volume=test-02
VolSessionId=2
VolSessionTime=1024128917
Volume=test-02
VolSessionId=1
VolSessionTime=1024132350
Volume=test-02
VolSessionId=1
VolSessionTime=1024380678

```

As a final example, assume that the initial Full save spanned two Volumes. The output from **query** might look like:

```

+-----+-----+-----+-----+-----+-----+
| JobId | StartTime | VolumeName | StartFile | VolSesId | VolSesTime |
+-----+-----+-----+-----+-----+-----+
| 242   | 2002-06-25 16:50 | File0003   | 0         | 1        | 1025016612 |
| 242   | 2002-06-25 16:50 | File0004   | 0         | 1        | 1025016612 |
| 243   | 2002-06-25 16:52 | File0005   | 0         | 2        | 1025016612 |
| 246   | 2002-06-25 19:19 | File0006   | 0         | 2        | 1025025494 |
+-----+-----+-----+-----+-----+-----+

```

and the following bootstrap file would restore those files:

```

Volume=File0003
VolSessionId=1
VolSessionTime=1025016612
Volume=File0004
VolSessionId=1
VolSessionTime=1025016612
Volume=File0005
VolSessionId=2
VolSessionTime=1025016612
Volume=File0006
VolSessionId=2
VolSessionTime=1025025494

```

28.2 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.

28.3 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). In that case, you can create a simple bootstrap file that consists of only the volume names. An example might be:

```
Volume="Vol1001"  
Volume="Vol1002"  
Volume="Vol1003"  
Volume="Vol1004"  
Volume="Vol1005"
```

28.4 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="Vol1001"  
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 29

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>.

29.1 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>.

29.2 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 ©2000-2009 Free Software Foundation Europe e.V.

Portions may be copyrighted by other people. These files are released under different licenses which are compatible with the Bacula GPLv2 license.

29.3 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.

29.4 Public Domain

Some of the Bacula code, or code that Bacula references, has been released to the public domain. E.g. md5.c, SQLite.

29.5 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. If you have made modifications to the Bacula source code that alter in any significant way the way the program functions, you may not distribute it using the Bacula name.

29.6 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 if you are submitting code, you should fill it out then sent to:

Kern Sibbald
Cotes-de-Montmoiret 9
1012 Lausanne
Switzerland

When you send in such a complete document, please notify me: kern at sibbald dot com.

29.7 Disclaimer

NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Chapter 30

GNU Free Documentation License

Version 1.2, November 2002

Copyright ©2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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. Secondly, 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

30.1 Table of Contents

- GNU GENERAL PUBLIC LICENSE
 - Preamble
 - TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
 - How to Apply These Terms to Your New Programs

30.2 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

30.3 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.

30.4 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

30.5 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.

Copyright notice above. Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

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.

30.6 Table of Contents

- GNU LESSER GENERAL PUBLIC LICENSE
 - Preamble
 - TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
 - How to Apply These Terms to Your New Libraries

30.7 GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

30.8 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.

30.9 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

30.10 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.

Copyright notice above. Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301
USA USA

Updated: 27 Nov 2000 paulv

Chapter 31

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 32

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 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.ossproject.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

Certain words and/or products are Copyrighted or Trademarked such as Windows (by Microsoft). Since they are numerous, and we are not necessarily aware of the details of each, we don't try to list them here. However, we acknowledge all such Copyrights and Trademarks, and if any copyright or trademark holder wishes a specific acknowledgment, notify us, and we will be happy to add it where appropriate.

32.1 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 33

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.

33.1 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`.

33.2 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.

33.3 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'

```

33.4 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.

33.5 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 34

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.

34.1 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.

34.2 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.

34.3 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
    |=====|
```

34.4 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.

34.5 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.

34.6 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.

34.7 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 CPath 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
```

34.8 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.

34.9 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.

34.10 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.

34.11 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 CPath 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

```

34.12 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.

34.13 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 CPath 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.

34.14 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.

34.15 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.

34.16 Using ssh to Secure the Communications

Please see the script **ssh-tunnel.sh** in the **examples** directory. It was contributed by Stephan Holl.

Chapter 35

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.

35.1 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-handler` 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
```

35.2 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)
```

35.3 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.

35.4 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).

General Index

- *JobLevel , 220
- *JobType , 220
- docdir configure option, 37
- htmldir configure option, 37
- plugindir configure option, 37
- MAJOR WARNING , 162
- Accurate Backup, 13
- ACL Updates, 16
- Actual Conf Files, 125
- Adapting Your mtx-changer script, 152
- Adding a Second Client , 74
- Additional Resources, 201
- Advantages , 135, 136
- Advantages of Bacula Over Other Backup Programs , 55
- Algorithm
 - New Volume, 104
 - Recycling , 104
- Allow Duplicate Jobs, 21
- Allow Higher Duplicates, 21
- Allow Mixed Priority, 31
- Alternate Disaster Recovery Suggestion for Win32 Systems, 200
- ANSI and IBM Tape Labels, 173
- Arguments
 - Command Line , 90
- Attribute Despooing, 36
- Attributes
 - Restoring Directory , 92
- Autochanger
 - Simulating Barcodes in your , 151
 - Using the , 153
- Autochanger Support , 141
- Autochangers
 - Supported, 157
 - Supported , 62
- Automated Disk Backup, 123
- Automatic Generation of Bootstrap Files , 222
- Automatic Pruning, 102
- Automatic Pruning and Recycling Example , 108
- Automatic Volume Labeling , 113
- Automatic Volume Recycling , 101
- Aware
 - FreeBSD Users Be , 62
- Backing Up the WinNT/XP/2K System State, 187
- Backing up to Multiple Disks , 117
- Backup
 - Simple One Tape , 135
- Backup Strategies , 135
- Bacula
 - Before Running , 65
 - Disaster Recovery Using, 191
 - What is , 1
 - Who Needs , 1
- Bacula Autochanger Interface , 155
- Bacula Bugs , 249
- Bacula Components or Services , 1
- Bacula Configuration , 4
- Bacula Copyright, Trademark, and Licenses, 225
- Bacula Events, 166
- Bacula Projects , 245
- Bacula Variables , 251
- baculoa-dir.conf
 - Modification for the Data Channel , 257
- Barcode Support , 154
- Bare Metal Recovery on Linux with a Rescue CD, 192
- Bare Metal Recovery using a LiveCD, 196
- Basic Volume Management, 111
- Bat Enhancements, 32
- Before Running Bacula , 65
- bextract handles Win32 non-portable data, 22
- Boot with your Rescue CDROM, 193
- Bootstrap Example, 223
- Bootstrap File , 219
- Bootstrap File Directive, 32
- Bootstrap File Format , 219
- Brief Tutorial , 65
- bscan, 223
 - bootstrap, 223
- bscan bootstrap, 223
- Bugs
 - Bacula , 249
 - Linux Problems or, 196
- Bugs and Other Considerations, 199
- Building Bacula with Encryption Support, 210
- Cancel Queued Duplicates, 22
- Cancel Running Duplicates, 22
- Catalog Format, 20
- CDROM
 - Bare Metal Recovery on Linux with a Rescue, 192
 - Boot with your Rescue, 193
- Certificate
 - Creating a Self-signed , 204, 261
 - Getting a CA Signed , 205, 262

- Certificates , 256
- Changing Cartridges , 150
- Channel
 - Encrypting the Control , 258
 - Securing the Data , 256
 - Starting and Testing the Control , 260
- Client
 - Adding a Second , 74
 - Using stunnel to Encrypt to a Second , 260
 - Win32 Specific File daemon Command Line Options, 188
- Client , 220
- Clients
 - Considerations for Multiple , 118
- Command
 - Console Restore, 83
 - Full Form of the Update Slots , 151
 - Restore, 83
- Command Line Arguments , 90
- Command Separator, 30
- Commands
 - File Selection , 94
 - Other Useful Console , 77
- Communications
 - Using ssh to Secure the , 262
- Communications Encryption, 203
- Communications Ports Used , 255
- Concurrent Disk Jobs, 114
- CONDITIONS
 - TERMS AND , 234, 239
- Config Files for stunnel to Encrypt the Control Channel , 259
- Configuration
 - Bacula , 4
 - Python, 165
- Connect Timeout, 33
- Considerations
 - Bugs and Other, 199
 - Important, 191
 - Windows Compatibility, 181
- Console Additions, 30
- Console Command
 - Python, 169
- Console Restore Command, 83
- Contents
 - Table of , 233, 238
- Control Channel Configuration , 259
- Conventions Used in this Document , 4
- Copy, 129
- Copy Jobs, 13
- Copyrights and Trademarks , 248
- Count, 220
- Creating a Pool , 79
- Creating a Self-signed Certificate , 204, 261
- Current Implementation Restrictions , 55
- Current State of Bacula , 53
- Daemon Command Line Options , 78
- Daemons
 - Starting the , 66

- Daily Tape Rotation , 136
- Daily, Weekly, Monthly Tape Usage Example , 106
- Data Encryption, 209
- Data Spooling , 161
- Data Spooling Directives , 161
- Database
 - Restoring, 96
 - Starting the , 66
- dbcheck enhancements, 37
- Dealing with Multiple Magazines , 150
- Dealing with Win32 Problems, 179
- Debug Daemon Output , 77
- Debugging Python Scripts, 169
- Decrypting with a Master Key, 210
- Design
 - Overall, 124
- Design Limitations or Restrictions , 56
- Details
 - Practical , 135, 137
- Details , 213
- Device Configuration Records , 144
- Devices
 - Multiple, 143
- devices
 - SCSI, 142
- Differential Max Wait Time, 34
- Differential Pool, 125
- Difficulties Connecting from the FD to the SD, 78
- Directives
 - Data Spooling , 161
 - DVD, 263, 265
 - DVD Edit Codes, 264
 - Pruning , 102
- Disadvantages , 135, 137
- Disaster
 - Preparing Solaris Before a, 198
- Disaster Recovery of Win32 Systems, 199
- Disaster Recovery Using Bacula, 191
- Disclaimer , 226
- Disk
 - Automated Backup, 123
- Disk Volumes, 111
- Disks
 - Backing up to Multiple , 117
- Document
 - Conventions Used in this , 4
- Domain
 - Public , 225
- Drives
 - Supported Tape , 61
 - Unsupported Tape , 62
- Duplicate Jobs, 21
- DVD Specific Director Directives , 265
- DVD Specific SD Directives , 263
- DVD Volumes, 263
- DVD Writing, 263
- Edit Codes for DVD Directives , 264
- Enable VSS, 183
- Encrypting the Control Channel , 258

- Encryption
 - Communications, 203
 - Data, 209
 - Starting and Testing the Data , 258
 - Transport, 203
- Encryption , 255
- Encryption Technical Details, 210
- Errors
 - Restore, 94
- Events, 166
- Example
 - Automatic Pruning and Recycling , 108
 - Bootstrap, 223
 - Daily Weekly Monthly Tape Usage , 106
 - Data Encryption Configuration File, 211
 - File Daemon Configuration File, 211
 - Python, 170
 - TLS Configuration Files, 205
 - Verify Configuration , 216
- Example , 115
- Example Configuration File , 148
- Example Data Encryption Configuration, 211
- Example Migration Jobs, 133
- Example Restore Job Resource , 94
- Example Scripts , 143
- Examples , 253
- Expansion
 - Variable , 251
- Extended Attributes, 17
- FD Version, 34
- FDL , 225
- Fiduciary License Agreement , 226
- File
 - Bootstrap , 219
 - Example Configuration , 148
- File Selection Commands , 94
- FileIndex, 220
- Filename
 - Selecting Files by , 88
- FileRegex, 220
- Files
 - Actual Conf, 125
 - Automatic Generation of Bootstrap , 222
 - Problems Restoring , 93
 - Restoring Your , 72
- Fills
 - When The Tape , 75
- Firewalls
 - Windows, 184
- Format
 - Bootstrap, 219
- Found
 - What To Do When Differences Are , 215
- FreeBSD Bare Metal Recovery, 197
- FreeBSD Issues , 152
- FreeBSD Users Be Aware , 62
- ftruncate for NFS Volumes, 33
- Full Form of the Update Slots Command , 151
- Full Pool, 124
- Full Syntax , 252
- Functionality
 - General , 251
- General, 191
- General , 83
- General Functionality , 251
- Generating Private/Public Encryption Keypairs, 211
- Getting a CA Signed Certificate , 205, 262
- GNU Free Documentation License, 227
- GNU GENERAL PUBLIC LICENSE , 233
- GNU General Public License , 233
- GNU LESSER GENERAL PUBLIC LICENSE , 238
- GNU Lesser General Public License , 238
- GPL , 225
- Have
 - Knowing What SCSI Devices You , 142
- How to Apply These Terms to Your New Libraries , 244
- How to Apply These Terms to Your New Programs , 237
- IgnoreDir, 23
- Implemented
 - What, 53
- Important Considerations, 191
- Important Migration Considerations, 132
- Incremental Max Wait Time, 34
- Incremental Pool, 125
- Installation, 175
- Interactions Between the Bacula Services, 8
- Interface
 - Bacula Autochanger , 155
- Issues
 - FreeBSD , 152
- Job
 - Running a , 68
- Job , 220
- JobId , 220
- Jobs
 - Querying or Starting Jobs, 66
- Key Concepts and Resource Records , 111
- Knowing What SCSI Devices You Have , 142
- Labeling
 - Automatic Volume , 113
 - Specifying Slots When , 149
- Labeling Volumes with the Console Program , 80
- Labeling Your Volumes , 79
- Labels
 - Tape, 173
- LGPL , 225
- libdbi Framework, 28
- Libraries
 - How to Apply These Terms to Your New , 244
- LICENSE

- GNU GENERAL PUBLIC , 233
- GNU LESSER GENERAL PUBLIC , 238
- License
 - GNU Free Documentation, 227
 - GNU General Public , 233
 - GNU Lesser General Public , 238
- Licenses
 - Bacula Copyright Trademark, 225
- Linux Problems or Bugs, 196
- list joblog, 30
- LiveCD
 - Bare Metal Recovery using a LiveCD, 196
- Magazines
 - Dealing with Multiple , 150
- Making Bacula Use a Single Tape, 106
- Management
 - Basic Volume, 111
- Manually Changing Tapes , 136
- Manually Recycling Volumes , 109
- Manually resetting the Permissions, 185
- Max Run Sched Time, 34
- Max Run Time directives, 34
- Max Wait Time, 34
- MaxConsoleConnections, 36
- MaxDiffInterval, 23
- MaxFullInterval, 23
- Microsoft Exchange Server 2003/2007 Plugin, 26
- Migration, 129
- Misc New Features, 31
- Modification of bacula-dir.conf for the Data Channel , 257
- Multi-drive Example Configuration File , 148
- Multiple Clients, 118
- Multiple Devices, 143
- New Features, 13
- New Volume Algorithm, 104
- Objects
 - Python, 166
- Options
 - Daemon Command Line , 78
- Other Points , 162, 265
- Other Useful Console Commands , 77
- Output
 - Debug Daemon , 77
- Overall Design, 124
- Permissions
 - Manually resetting the, 185
- Picture , 256
- Plugin, 24
- Plugin Directory, 24
- Plugin Options, 24
- Plugin Options ACL, 24
- Points
 - Other , 162, 265
- Pool
 - Creating a , 79
 - Differential, 125
 - Full, 124
 - Incremental, 125
 - Pool Options to Limit the Volume Usage , 112
 - Post Win32 Installation, 179
 - Practical Details , 135, 137
 - Preamble , 233, 238
 - Preparing Solaris Before a Disaster, 198
 - Problem, 123
 - Problems
 - VSS, 183
 - Windows Backup, 185
 - Windows Ownership and Permissions, 185
 - Windows Restore, 184
 - Problems Restoring Files , 93
 - Program
 - Labeling Volumes with the Console , 80
 - Quitting the Console , 74
 - Programs
 - Advantages of Bacula Over Other Backup , 55
 - How to Apply These Terms to Your New , 237
 - Projects
 - Bacula , 245
 - Pruning
 - Automatic, 102
 - Pruning Directives , 102
 - Public Domain , 225
 - Python Configuration, 165
 - Python Console Command, 169
 - Python Example, 170
 - Python Objects, 166
 - Python Scripting, 165
 - Querying or starting Jobs, 66
 - Quick Start , 5
 - Quitting the Console Program , 74
 - Records
 - Device Configuration , 144
 - Key Concepts and Resource , 111
 - Recovery
 - Bare Metal Recovery using a LiveCD, 196
 - Disaster Recovery, 191
 - FreeBSD Bare Metal, 197
 - Solaris Bare Metal, 198
 - Windows Disaster, 184
 - Recycle Pool, 33
 - Recycle Status , 105
 - Recycling
 - Automatic Volume , 101
 - Restricting the Number of Volumes and Recycling, 113
 - Recycling Algorithm , 104
 - Requirements, 192
 - System , 57
 - Rescue
 - Bare Metal Recovery using a LiveCD, 196
 - Disaster Recovery, 191
 - FreeBSD Bare Metal, 197
 - Resetting Directory and File Ownership and Permissions on Win32 Systems, 199

- Resource
 - Example Restore Job , 94
- Resources
 - Additional, 201
- Restore Command, 83
- Restore Directories, 85
- Restore Errors, 94
- Restore menu, 9
- Restoring a Client System, 192
- Restoring a Server, 195
- Restoring Directory Attributes , 92
- Restoring Files Can Be Slow , 93
- Restoring on Windows , 92
- Restoring to a Running System, 200
- Restoring When Things Go Wrong , 96
- Restoring Your Database, 96
- Restoring Your Files , 72
- Restricting the Number of Volumes and Recycling, 113
- Restrictions
 - Current Implementation , 55
 - Design Limitations or , 56
- Rotation
 - Daily Tape , 136
- Running a Job , 68
- Running the Verify , 214
- RunScript Enhancements, 33
- ScratchPool, 36
- Scripting
 - Python, 165
- Scripts
 - Example , 143
- SCSI devices, 142
- SD
 - Difficulties Connecting from the FD to the SD, 78
- Securing the Data Channel , 256
- Security
 - Using Bacula to Improve Computer , 213
- Selecting Files by Filename , 88
- Semantics , 253
- Server
 - Restoring a, 195
- Services
 - Bacula Components or , 1
 - Interactions Between the Bacula, 8
- Shared objects, 18
- Shutting down Windows Systems, 189
- Simple One Tape Backup , 135
- Simulating Barcodes in your Autochanger , 151
- Slot , 220
- Slots , 143
- Slow
 - Restoring Files Can Be , 93
- Solaris Bare Metal Recovery, 198
- Solution, 123
- Source Address, 10
- Specifications
 - Tape, 62
- Specifying Slots When Labeling , 149
- Spooling
 - Data , 161
- SpoolSize, 36
- Start
 - Quick , 5
- Starting and Testing the Control Channel , 260
- Starting and Testing the Data Encryption , 258
- Starting the Daemons , 66
- Starting the Database , 66
- State
 - Backing Up the WinNT/XP/2K System, 187
- State File, 22
- Static linking, 18
- Statistics, 163
- Statistics Enhancements, 35
- Status
 - Recycle , 105
- Status Enhancements, 33
- StatusSlots, 30
- Steps to Take Before Disaster Strikes, 191
- Strategies
 - Backup , 135
- Stream , 220
- Strikes
 - Steps to Take Before Disaster, 191
- Stunnel Configuration for the Data Channel , 257
- Support
 - Autochanger , 141
 - Barcode , 154
- Supported Autochanger Models, 157
- Supported Autochangers , 62
- Supported Operating Systems , 59
- Supported Tape Drives , 61
- Syntax
 - Full , 252
- System
 - Restoring a Client, 192
 - Restoring to a Running, 200
- System Requirements , 57
- Systems
 - Alternate Disaster Recovery Suggestion for Win32, 200
 - Disaster Recovery of Win32, 199
 - Resetting Directory and File Ownership and Permissions on Win32, 199
 - Shutting down Windows, 189
 - Supported Operating , 59
- Table of Contents , 233, 238
- Tape
 - Making Bacula Use a Single, 106
- Tape Specifications, 62
- Tapes
 - Manually Changing , 136
- Terminology , 5
- TERMS AND CONDITIONS , 234, 239
- Testing the Autochanger , 152
- Thanks , 247
- The bpipe Plugin, 25

- TLS, 203
- TLS – Communications Encryption, 203
- TLS Authentication, 22
- TLS Configuration Files, 205
- Trademark , 226
- Trademarks
 - Copyrights and , 248
- Transport Encryption, 203
- Tutorial
 - Brief , 65
- Unicode, 188
- Uninstalling Bacula on Win32, 179
- Unsupported Tape Drives , 62
- Upgrading, 177
- Usage
 - Pool Options to Limit the Volume , 112
 - Windows Port, 184
- Used
 - Communications Ports , 255
- Using Bacula to Improve Computer Security , 213
- Using File Relocation, 91
- Using Pools to Manage Volumes, 123
- Using ssh to Secure the Communications , 262
- Using Stunnel to Encrypt Communications to Clients , 255
- Using stunnel to Encrypt to a Second Client , 260
- Using the Autochanger , 153
- Variable Expansion , 251
- Variables
 - Bacula , 251
- Vbackup, 18
- VerId, 37
- Verify
 - Running the , 214
- Verify Configuration Example , 216
- Virtual Backup, 18
- Virtual Tape Emulation, 32
- VolBlock, 220
- VolFile, 220
- VolSessionId , 220
- VolSessionTime , 220
- Volume , 219
- Volume Shadow Copy Service, 182
- Volumes
 - DVD, 263
 - Labeling Your , 79
 - Manually Recycling , 109
 - Using Pools to Manage, 123
- VSS, 182
- VSS Problems, 183
- WARNING
 - MAJOR , 162
- What Bacula is Not, 7
- What is Bacula? , 1
- What is Implemented, 53
- What To Do When Differences Are Found , 215
- When The Tape Fills , 75
- Who Needs Bacula? , 1
- Win32
 - Dealing with Problems, 179
 - Installation, 175
 - Post Installation, 179
 - Uninstalling Bacula, 179
- Win32 Path Length Restriction, 188
- Win32 Specific File daemon Command Line Options, 188
- Win64 Client, 20
- Windows
 - Considerations for Filename Specifications, 188
 - Restoring on , 92
- Windows Backup Problems, 185
- Windows Compatibility Considerations, 181
- Windows Disaster Recovery, 184
- Windows Firewalls, 184
- Windows Ownership and Permissions Problems, 185
- Windows Port Usage, 184
- Windows Restore Problems, 184
- Windows Version of Bacula, 175
- Writing DVDs, 263

Director Index

count , 95
Counters , 251

dir , 95
done , 95

Enable VSS, 183
Environment Variables , 252
estimate , 95

find, 95

Internal Variables , 251

JobStart, 166

mark, 95

pwd , 95

unmark , 95

Write Part After Job , 265

File Daemon Index

- *Archive , 6
- /about, 188
- /events, 188
- /help, 188
- /install, 188
- /kill, 188
- /remove, 188
- /run, 188
- /service, 188
- /status, 188
- , 183
- a name , 6
- Administrator , 5
- Backup , 5
- Bootstrap File , 5
- Catalog , 5
- Client , 5
- Console , 5
- Daemon , 5
- Differential , 5
- Directive , 5
- Director , 5
- exit , 95
- File Attributes , 5
- File Daemon , 5
- help , 95
- Incremental , 6
- lsmark, 95
- Monitor , 6
- quit , 95
- Recycle , 103
- Resource , 6
- Restore , 6
- Retention Period , 7
- Schedule , 6
- Service , 6
- Storage Coordinates , 6
- Storage Daemon , 6
- VSS Problems, 183

Storage Daemon Index

-c <file> , 78

-d nn , 78

Autochanger , 144

Autochanger Resource, 147

Changer Command , 144, 147

Changer Device, 147

Changer Device , 144

Drive Index , 145

Free Space Command , 264

Maximum Changer Wait , 145

Mount Command, 263

Mount Point, 263

mount storage , 77

mtx-changer list, 152

mtx-changer load, 152

mtx-changer loaded, 152

mtx-changer slots, 152

mtx-changer unload, 152

Name, 147

quit , 77

Requires Mount , 263

Resource

 Autochanger, 147

Scan , 7

Session , 6

Unmount Command, 264

Verify , 6

Volume , 7

Write Part Command , 264

Console Index

AutoPrune , 103

list files jobid , 77

list jobid , 77

list jobmedia , 77

list jobs , 77

list jobtotals , 77

list media , 77

list pools , 77

messages , 77

status , 77

status dir , 77

status jobid , 77

unmount storage , 77

Volume Retention, 103