

Cedar Backup Software Manual

Kenneth J. Pronovici

Cedar Backup Software Manual

by Kenneth J. Pronovici

Copyright © 2005 Kenneth J. Pronovici

This work is free; you can redistribute it and/or modify it under the terms of the GNU General Public License (the "GPL"), Version 2, as published by the Free Software Foundation.

For the purposes of the GPL, the "preferred form of modification" for this work is the original Docbook XML text files. If you choose to distribute this work in a compiled form (i.e. if you distribute HTML, PDF or Postscript documents based on the original Docbook XML text files), you must also consider image files to be "source code" if those images are required in order to construct a complete and readable compiled version of the work.

This work 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.

Copies of the GNU General Public License are available from the Free Software Foundation website, <http://www.gnu.org/>. You may also write the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Table of Contents

Preface	vi
Purpose	vi
Audience	vi
Conventions Used in This Book	vi
Typographic Conventions	vi
Icons	vi
Organization of This Manual	vii
Acknowledgments	vii
1. Introduction	1
What is Cedar Backup?	1
How to Get Support	1
History	2
2. Basic Concepts	4
General Architecture	4
Cedar Backup Pools	4
The Backup Process	4
The Collect Action	5
The Stage Action	5
The Store Action	6
The Purge Action	6
The All Action	6
The Validate Action	6
The Rebuild Action	7
Coordination between Master and Clients	7
Media and Device Types	7
Incremental Backups	8
Extensions	9
3. Installation	10
Background	10
Installing on a Debian System	10
Installing from Source	11
Installing Dependencies	11
Installing the Source Package	12
4. Configuration	13
Overview	13
Command Line Interface	13
Syntax	13
Switches	14
Actions	15
Configuration File Format	15
Sample Configuration File	16
Reference Configuration	17
Options Configuration	18
Collect Configuration	19
Stage Configuration	23
Store Configuration	26
Purge Configuration	28
Extensions Configuration	29
Setting up a Pool of One	30

Step 1: Make sure email works.	30
Step 2: Configure your CD-R or CD-RW drive.	31
Step 3: Configure your backup user.	31
Step 4: Create your backup tree.	31
Step 5: Modify the backup cron jobs.	32
Step 6: Create the Cedar Backup configuration file.	33
Step 7: Validate the Cedar Backup configuration file.	33
Step 8: Test your backup.	33
Setting up a Client Peer Node	33
Step 1: Make sure email works.	34
Step 2: Configure the master in your backup pool.	34
Step 3: Configure your backup user.	34
Step 4: Create your backup tree.	35
Step 5: Modify the backup cron jobs.	36
Step 6: Create the Cedar Backup configuration file.	36
Step 7: Validate the Cedar Backup configuration file.	37
Step 8: Test your backup.	37
Setting up a Master Peer Node	37
Step 1: Make sure email works.	37
Step 2: Configure your CD-R or CD-RW drive.	38
Step 3: Configure your backup user.	38
Step 4: Create your backup tree.	39
Step 5: Modify the backup cron jobs.	39
Step 6: Create the Cedar Backup configuration file.	40
Step 7: Validate the Cedar Backup configuration file.	40
Step 8: Test connectivity to client machines.	40
Step 9: Test your backup.	41
5. Official Extensions	42
System Information Extension	42
Subversion Extension	42
MySQL Extension	44
A. Extension Architecture Interface	47
B. Dependencies	49
C. Copyright	52

Preface

Purpose

This software manual has been written to document the 2.0 series of Cedar Backup, originally released in early 2005.

Audience

This manual has been written for computer-literate administrators who need to use and configure Cedar Backup on their Linux system. The examples in this manual assume the reader is relatively comfortable with Unix and command-line interfaces.

Conventions Used in This Book

This section covers the various conventions used in this manual.

Typographic Conventions

Term

Used for first use of important terms.

Command

Used for commands, command output, and switches

Replaceable

Used for replaceable items in code and text

Filenames

Used for file and directory names

Icons



Note

This icon designates a note relating to the surrounding text.



Tip

This icon designates a helpful tip relating to the surrounding text.



Warning

This icon designates a warning relating to the surrounding text.

Organization of This Manual

Chapter 1, *Introduction*

Provides some background about how Cedar Backup came to be, its history, some general information about what needs it is intended to meet, etc.

Chapter 2, *Basic Concepts*

Discusses the basic concepts of a Cedar Backup infrastructure, and specifies terms used throughout the rest of the manual.

Chapter 3, *Installation*

Explains how to install the Cedar Backup package either from the Python source distribution or from the Debian package.

Chapter 4, *Configuration*

Provides detailed information about how to configure Cedar Backup.

Chapter 5, *Official Extensions*

Describes each of the officially-supported Cedar Backup extensions.

Appendix A, *Extension Architecture Interface*

Specifies the Cedar Backup extension architecture interface, through which third party developers can write extensions to Cedar Backup.

Appendix B, *Dependencies*

Provides some additional information about the packages which Cedar Backup relies on, including information about how to find documentation and packages on non-Debian systems.

Acknowledgments

The structure of this manual and some of the basic boilerplate has been taken from the book Version Control with Subversion [<http://svnbook.red-bean.com/>]. Many thanks to the authors (and O'Reilly) for making this excellent reference available under a free and open license.

There are not very many Cedar Backup users today, but almost all of them have contributed in some way to the documentation in this manual, either by asking questions, making suggestions or finding bugs. I'm glad to have them as users, and I hope that this new release meets their needs even better than the previous release.

My wife Julie puts up with a lot. It's sometimes not easy to live with someone who hacks on open source code in his free time — even when you're a pretty good engineer yourself, like she is. First, she managed to live with a dual-boot Debian and Windoze machine; then she managed to get used to IceWM rather than a prettier desktop; and eventually she even managed to cope with **vim** when she needed to. Now, even after all that, she has graciously volunteered to edit this manual. I much appreciate her skill with a red pen.

Chapter 1. Introduction

“Only wimps use tape backup: real men just upload their important stuff on ftp, and let the rest of the world mirror it.”— Linus Torvalds, at the release of Linux 2.0.8 in July of 1996.

What is Cedar Backup?

Cedar Backup is a Python package that supports secure backups of files on local and remote hosts to CD-R or CD-RW media. Cedar Backup also includes extensions that understand how to back up MySQL databases and Subversion repositories, and it can be easily extended to support other data sources, as well.

The package is focused around weekly backups to a single disc, with the expectation that the disc will be changed or overwritten at the beginning of each week. If your hardware is new enough, Cedar Backup can write multisession discs, allowing you to add to a disc in a daily fashion. Directories are backed up using **tar** and may be compressed using **gzip** or **bzip2**.

There are many different backup software implementations out there in the free-and open-source world. Cedar Backup aims to fill a niche: it aims to be a good fit for people who need to back up a limited amount of important data to CD-R or CD-RW on a regular basis. Cedar Backup isn't for you if you want to back up your MP3 collection every night, or if you want to back up a few hundred machines. However, if you administer a small set machines and you want to run daily incremental backups for things like system configuration, current email, small web sites, or a CVS repository, then Cedar Backup is probably worth your time.

Cedar Backup has been developed on a Debian GNU/Linux system and is currently supported only on Debian and other Linux systems. However, since it is written in portable Python, it should in theory run without too many problems on other UNIX-like systems which have a working version of the **cdrecord** and **mkisofs** utilities.

How to Get Support

Cedar Backup is open source software that is provided to you at no cost. It is provided with no warranty, not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. However, that said, someone can usually help you solve whatever problems you might see.

If you experience a problem, your best bet is to write the Cedar Backup Users mailing list.¹ This is a public list for all Cedar Backup users. If you write to this list, you might get help from me, or from some other user who has experienced the same thing you have.

If you know that the problem you have found constitutes a bug, or if you would like to make an enhancement request, then feel free to file a bug report in the Cedar Solutions Bug Tracking System.²

If you are not comfortable discussing your problem in public or listing it in a public database, or if you need to send along information that you do not want made public, then you can write <support@cedar-solutions.com>. That mail will go directly to me or to someone else who can

¹See <http://cedar-solutions.com/listarchives/>.

²See <http://cedar-solutions.com/bugzilla/>.

help you. If you write the support address about a bug, a “scrubbed” bug report will eventually end up in the public bug database anyway, so if at all possible you should use the public reporting mechanisms. One of the strengths of the open-source software development model is its transparency.

Regardless of how you report your problem, please try to provide as much information as possible about the behavior you observed and the environment in which the problem behavior occurred.³

In particular, you should provide: the version of Cedar Backup that you are using; how you installed Cedar Backup (i.e. Debian package, source package, etc.); the exact command line that you executed; any error messages you received, including Python stack traces (if any); and relevant sections of the Cedar Backup log. It would be even better if you could describe exactly how to reproduce the problem, for instance by including your entire configuration file and/or specific information about your system that might relate to the problem. However, please do *not* provide huge sections of debugging logs unless you are sure they are relevant or unless someone asks for them.



Tip

Sometimes, the error that Cedar Backup displays can be rather cryptic. This is because under internal error conditions, the text related to an exception might get propagated all of the way up to the user interface. If the message you receive doesn't make much sense, or if you suspect that it results from an internal error, you might want to re-run Cedar Backup with the `--stack` option. This forces Cedar Backup to dump the entire Python stack trace associated with the error, rather than just printing the last message it received. This is good information to include along with a bug report, as well.

History

Cedar Backup began life in late 2000 as a set of Perl scripts called kbackup. These scripts met an immediate need (which was to back up skyjammer.com and some personal machines) but proved to be unstable, overly verbose and rather difficult to maintain.

In early 2002, work began on a rewrite of kbackup. The goal was to address many of the shortcomings of the original application, as well as to clean up the code and make it available to the general public. While doing research related to code I could borrow or base the rewrite on, I discovered that there was already an existing backup package with the name kbackup, so I decided to change the name to Cedar Backup instead.

Because I had become fed up with the prospect of maintaining a large volume of Perl code, I decided to abandon that language in favor of Python.⁴ At the time, I chose Python mostly because I was interested in learning it, but in retrospect it turned out to be a very good decision. Python has almost all of the strengths of Perl, but few of its inherent weaknesses (primarily, Python code often ends up being much more readable than Perl code).

Around this same time, skyjammer.com and cedar-solutions.com were converted to run Debian GNU/Linux (potato)⁵ and I entered the Debian new maintainer queue, so I also made it a goal to implement Debian packages along with a Python source distribution for the new release.

Version 1.0 of Cedar Backup was released in June of 2002. We immediately began using it to back up

³See Simon Tatham's excellent bug reporting tutorial: <http://www.chiark.greenend.org.uk/~sgtatham/bugs.html> .

⁴See <http://www.python.org/> .

⁵Debian's stable releases are named after characters in the Toy Story movie.

skyjammer.com and cedar-solutions.com, where it proved to be much more stable than the original code. Since then, we have continued to use Cedar Backup for those sites, and Cedar Backup has picked up a handful of other users who have occasionally reported bugs or requested minor enhancements.

In the meantime, I continued to improve as a Python programmer and also started doing a significant amount of professional development in Java. It soon became obvious that the internal structure of Cedar Backup 1.0, while much better than kbackup, still left something to be desired. In November 2003, I began an attempt at cleaning up the codebase. I converted all of the internal documentation to use Epydoc,⁶ and updated the code to use the newly-released Python logging package⁷ after having a good experience with Java's log4j. However, I was still not satisfied with the code, which did not lend itself to the automated regression testing I had used when working with junit in my Java code.

So, rather than releasing the cleaned-up code, I instead began another ground-up rewrite in May 2004. With this rewrite, I applied everything I had learned from other Java and Python projects I had undertaken over the last few years. I structured the code to take advantage of Python's unique ability to blend procedural code with object-oriented code, and I made automated unit testing a primary requirement. The result is the 2.0 release, which is cleaner, more compact, better focused, and better documented than any release before it. Utility code is less application-specific, and is now usable as a general-purpose library. The 2.0 release also includes a complete regression test suite of over 1900 tests, which will help to ensure that quality is maintained as development continues into the future.⁸

⁶Epydoc is a Python code documentation tool. See <http://epydoc.sourceforge.net/>.

⁷See <http://docs.python.org/lib/module-logging.html>.

⁸Tests are implemented using Python's unit test framework. See <http://docs.python.org/lib/module-unittest.html>.

Chapter 2. Basic Concepts

General Architecture

Cedar Backup is architected as a Python package (library) and single executable (a Python script). The Python package provides both application-specific code and general utilities which can be used by programs other than Cedar Backup. It also includes modules that can be used by third parties to extend Cedar Backup or provide related functionality.

The **cback** script is designed to run as root, since otherwise it's difficult to back up system directories or write to the CD-R/CD-RW device. However, pains are taken to use the backup user's effective user id (specified in configuration) when appropriate. Note: this does not mean that **cback** runs *setuid*¹ or *setgid*. However, all files on disk will be owned by the backup user, and all rsh-based network connections will take place as the backup user.

Cedar Backup provides no facility for restoring backups. It is assumed that the user can copy tarfiles off disc and use them to restore missing files as needed.

The **cback** script is configured via command-line options and an XML configuration file on disk. The configuration file is normally stored in `/etc/cback.conf`, but this path can be overridden at runtime. See Chapter 4, *Configuration* for more information on how Cedar Backup is configured.



Warning

You should be aware that backups to CD media can probably be read by any user which has permissions to mount the CD writer. If you intend to leave the backup disc in the drive at all times, you may want to consider this when setting up device permissions on your machine.

Cedar Backup Pools

There are two kinds of machines in a Cedar Backup pool. One machine (the *master*) has a CD-R or CD-RW drive on it and writes the backup to disc. The others (*clients*) collect data to be written to disc by the master. Collectively, the master and client machines in a pool are called *peer machines*.

Cedar Backup has been designed primarily for situations where there is a single master and a set of other clients that the master interacts with. However, it will just as easily work for a single machine (a backup pool of one) and in fact more users seem to use it like this than any other way.

The Backup Process

The Cedar Backup backup process is structured in terms of a set of decoupled actions which execute independently (based on a schedule in **cron**) rather than through some highly coordinated flow of control.

¹See <http://en.wikipedia.org/wiki/Setuid>

This design decision has both positive and negative consequences. On the one hand, the code is much simpler and can choose to simply abort or log an error if its expectations are not met. On the other hand, the administrator must coordinate the various actions during initial set-up. See the section called “Coordination between Master and Clients” (later in this chapter) for more information on this subject.

A standard backup run consists of four steps (actions), some of which execute on the master machine, and some of which execute on one or more client machines. These actions are: *collect*, *stage*, *store* and *purge*.

In general, more than one action may be specified on the command-line. If more than one action is specified, then actions will be taken in a sensible order (generally collect, stage, store, purge). A special *all* action is also allowed, which implies all of the standard actions in the same sensible order.

The **cback** command also supports several actions that are not part of the standard backup run and cannot be executed along with any other actions. These actions are *validate* and *rebuild*. All of the various actions are discussed further below.



Note

See Chapter 4, *Configuration* for more information on how a backup run is configured.

The Collect Action

The collect action is the first action in a standard backup run. It executes both master and client nodes. Based on configuration, this action traverses the peer's filesystem and gathers files to be backed up. Each configured high-level directory is collected up into its own **tar** file in the *collect directory*. The tarfiles can either be uncompressed (*.tar*) or compressed with either **gzip** (*.tar.gz*) or **bzip2** (*.tar.bz2*).

There are three supported collect modes: *daily*, *weekly* and *incremental*. Directories configured for daily backups are backed up every day. Directories configured for weekly backups are backed up on the first day of the week. Directories configured for incremental backups are traversed every day, but only the files which have changed (based on a saved-off *SHA hash*) are actually backed up.

Collect configuration also allows for a variety of ways to filter files and directories out of the backup. For instance, administrators can configure an *ignore indicator file*² or specify absolute paths or filename patterns³ to be excluded.

The Stage Action

The stage action is the second action in a standard backup run. It executes on the master peer node. The master works down the list of peers in its backup pool and stages (copies) the collected backup files from each of them into a daily staging directory by peer name.

Local and remote client peers are treated differently. Local peer collect directories are assumed to be accessible via normal copy commands (i.e. on a mounted filesystem) while remote peer collect directories are accessed via an *RSH-compatible* command such as **ssh**.

If a given peer is not ready to be staged, the stage process will log an error, abort the backup for that

²Analogous to *.cvsignore* in CVS

³In terms of Python regular expressions

peer, and then move on to its other peers. This way, one broken peer cannot break a backup for other peers which are up and running.



Note

Directories “collected” by another process can be staged by Cedar Backup. If the file `cback.collect` exists in a collect directory when the stage action is taken, then that directory will be staged. Just beware that when staged, everything for a given host will reside in the same directory — so be careful to avoid namespace clash.

The Store Action

The store action is the third action in a standard backup run. It executes on the master peer node. The master machine determines the location of the current staging directory, and then writes the contents of that staging directory to disc. After the contents of the directory have been written to disc, an optional validation step ensures that the write was successful.

If the backup is running on the first day of the week, if the drive does not support multisession discs, or if the `--full` option is passed to the **cback** command, the disc will be rebuilt from scratch. Otherwise, a new ISO session will be added to the disc each day the backup runs.

The Purge Action

The purge action is the fourth and final action in a standard backup run. It executes both on the master and client peer nodes. Configuration specifies how long to retain files in certain directories, and older files and empty directories are purged.

Typically, collect directories are purged daily, and stage directories are purged weekly or slightly less often (if a disc gets corrupted, older backups may still be available on the master). Some users also choose to purge the configured working directory (which is used for temporary files) to eliminate any leftover files which might have resulted from changes to configuration.

The All Action

The all action is a pseudo-action which causes all of the actions in a standard backup run to be executed together in order. It cannot be combined with any other actions on the command line.

Extensions can be executed as part of the all action. If you need to execute an extended action, you must specify the other actions you want to run individually on the command line.

The all action does not have its own configuration. Instead, it relies on the individual configuration sections for all of the other actions.

The Validate Action

The validate action is used to validate configuration on a particular peer node, either master or client. It cannot be combined with any other actions on the command line.

The validate action checks that the configuration file can be found, that the configuration file is valid, and that certain portions of the configuration file make sense (for instance, making sure that specified

users exist, directories are readable and writable as necessary, etc.).

The Rebuild Action

The rebuild action is an exception-handling action that is executed independent of a standard backup run. It cannot be combined with any outside actions on the command line.

The rebuild action attempts to rebuild “this week's” disc from any remaining unpurged staging directories. Typically, it is used to make a copy of a backup, replace lost or damaged media, or to switch to new media mid-week for some other reason.

To decide what data to write to disc again, the rebuild action looks back and finds first day of the current week. Then, it finds any remaining staging directories between that date and the current date. If any staging directories are found, they are all written to disc in one big ISO session.

The rebuild action does not have its own configuration. It relies on configuration for other other actions, especially the store action.

Coordination between Master and Clients

Unless you are using Cedar Backup to manage a “pool of one”, you will need to set up some coordination between your clients and master to make everything work properly. This coordination isn't difficult — it mostly consists of making sure that operations happen in the right order — but some users are suprised that it is required and want to know why Cedar Backup can't just “take care of it for me”.

Essentially, each client must finish collecting all of its data before the master begins staging it, and the master must finish staging data from a client before that client purges its collected data. Administrators may need to experiment with the time between the collect and purge entries so that the master has enough time to stage data before it is purged.

Some users are initially surprised that Cedar Backup does not manage this coordination itself. The reason is, this coordination step usually doesn't take a lot of effort and is only imposed on the user at configuration time. However, to accomplish the same thing *dynamically* in code would add quite a bit of complexity to Cedar Backup. This code would be difficult to test and would initially be somewhat error-prone, at least until I worked out all of the kinks. Given that the current architecture has been proven to work well, I don't think that it is worth adding complexity to the code just to simplify the initial set-up process.⁴

Media and Device Types

Cedar Backup is focused around writing backups to CD-R or CD-RW media using a standard SCSI or IDE CD writer. In Cedar Backup terms, the disc itself is referred to as the *media*, and the CD-R or CD-RW drive is referred to as the *device* or sometimes the *backup device*.⁵

When using a new enough backup device, a new “multisession” ISO image⁶ is written to the media on the first day of the week, and then additional multisession images are added to the media each day that

⁴Feel free to write me or the user mailing list if you disagree *and* can come up with a straightforward implementation which can be easily verified and maintained.

⁵The backup device I develop against is a fairly old Sony CRX140E 4X CD-RW drive.

Cedar Backup runs. This way, the media is complete and usable at the end of every backup run, but a single disc can be used all week long. If your backup device does not support multisession images, then a new ISO image will be written to the media each time Cedar Backup runs (and you should probably confine yourself to the “daily” backup mode to avoid losing data).

Cedar Backup currently supports four different kinds of media:

cdr-74

74-minute non-rewritable media

cdrw-74

74-minute rewritable media

cdr-80

80-minute non-rewritable media

cdrw-80

80-minute rewritable media

I have chosen to support just these four types of media because they seem to be the most “standard” of the various types commonly sold in the U.S. today (early 2005). If you regularly use an unsupported media type and would like Cedar Backup to support it, send me information about the capacity of the media in megabytes (MB) and whether it is rewritable.

Future versions of Cedar Backup may support writable DVDs.⁷ However, this cannot happen unless I can get access to hardware for development and testing, or unless someone else is willing to do research and test code on my behalf. If you would like to see support for DVDs in Cedar Backup and can offer some help, please write the Cedar Backup Users mailing list.⁸

Incremental Backups

Cedar Backup supports three different kinds of backups for individual collect directories. These are *daily*, *weekly* and *incremental* backups. Directories using the daily mode are backed up every day. Directories using the weekly mode are only backed up on the first day of the week, or when the `--full` option is used. Directories using the incremental mode are always backed up on the first day of the week (like a weekly backup), but after that only the files which have changed are actually backed up on a daily basis.

In Cedar Backup, incremental backups are not based on date, but are instead based on saved checksums, one for each backed-up file. When a full backup is run, Cedar Backup gathers a checksum value⁹ for each backed-up file. The next time an incremental backup is run, Cedar Backup checks its list of file/checksum pairs for each file that might be backed up. If the file's checksum value does not match the

⁶An *ISO image* is the standard way of creating a filesystem to be copied to a CD. It is essentially a “filesystem-within-a-file” and most Linux systems can actually mount ISO image files just like hard drives, floppy disks or actual CDs. See Wikipedia for more information: http://en.wikipedia.org/wiki/ISO_image.

⁷It would just require a new `DvdWriter` class in `writer.py` as well as some minor changes to configuration code. All writer-related access is through an abstract interface, so once the new writer is implemented, the rest of the code will be able to use it without any changes.

⁸See <http://cedar-solutions.com/listarchives/>.

⁹The checksum is actually an *SHA cryptographic hash*. See Wikipedia for more information: <http://en.wikipedia.org/wiki/SHA-1>.

saved value, or if the file does not appear in the list of file/checksum pairs, then it will be backed up and a new checksum value will be placed into the list. Otherwise, the file will be ignored and the checksum value will be left unchanged.

Cedar Backup stores the file/checksum pairs in `.sha` files in its working directory, one file per configured collect directory. The mappings in these files are reset at the start of the week or when the `--full` option is used. Because these files are used for an entire week, you should never purge the working directory more frequently than once per week.

Extensions

Imagine that there is a third party developer who understands how to back up a certain kind of database repository. This third party might want to integrate his or her specialized backup into the Cedar Backup process, perhaps thinking of the database backup as a sort of “collect” step.

Prior to Cedar Backup 2.0, any such integration would have been completely independent of Cedar Backup itself. The “external” backup functionality would have had to maintain its own configuration and would not have had access to any Cedar Backup configuration.

Starting with version 2.0, Cedar Backup allows *extensions* to the backup process. An extension is an action that isn't part of the standard backup process, (i.e. not collect, stage, store or purge) but can be executed by Cedar Backup when properly configured.

Extension authors implement an “action process” function with a certain interface, and are allowed to add their own sections to the Cedar Backup configuration file, so that all backup configuration can be centralized. Then, the action process function is associated with an action name (i.e. “database”) which can be executed from the **cback** command line like any other action.

It is our hope that as use of Cedar Backup 2.0 grows, users will contribute their own extensions back to the community. Well-written general-purpose extensions will be accepted into the official codebase.



Note

Users should see Chapter 4, *Configuration* for more information on how extensions are configured, and Chapter 5, *Official Extensions* for details on all of the officially-supported extensions.

Developers may be interested in Appendix A, *Extension Architecture Interface*.

Chapter 3. Installation

Background

There are two different ways to install Cedar Backup. The easiest way is to install the pre-built Debian packages. This method is painless and ensures that all of the correct dependencies are available, etc.

If you are running a Linux distribution other than Debian, then you must use the Python source distribution to install Cedar Backup. When using this method, you need to manage all of the dependencies yourself.

Non-Linux Platforms

Currently, Cedar Backup is only “officially” supported on Linux platforms. However, it should also run on other UNIX-like systems where the **mkisofs** and **cdrecord** commands are available.

If you would like to use Cedar Backup on a non-Linux system, you should install the Python source distribution along with all of the indicated dependencies. Then, please report back to the Cedar Backup Users mailing list ¹ with information about your platform and any problems you encountered.

Installing on a Debian System

Not an “Official” Debian Package

Currently, Cedar Backup is not part of any official Debian release. While I am a Debian developer, I have chosen to keep the package out of the Debian archive until it demonstrates enough popularity to belong there. There is no point in putting it in Debian for three users.

The easiest way to install Cedar Backup onto a Debian system is by using a tool such as **apt-get** or **aptitude**.

First, add the Cedar Solutions APT data source to your `/etc/apt/sources.list` file. ² Then, execute:

```
$ apt-get update
$ apt-get install cedar-backup2
```

If you would prefer, you can also download the `.deb` files and install them by hand with a tool such as **dpkg**. You can find a link to the `.deb` files on the Cedar Solutions website. ³

¹See <http://cedar-solutions.com/listarchives/>.

²See <http://cedar-solutions.com/debian.html>.

³See <http://cedar-solutions.com/software.html>.

In either case, once the package has been installed, you can proceed to configuration as described in Chapter 4, *Configuration*.



Note

The Debian package-management tools must generally be run as root. It is safe to install Cedar Backup to a non-standard location and run it as a non-root user. However, to do this, you must install the source distribution instead of the Debian package.

Installing from Source

On platforms other than Debian, Cedar Backup is installed from a Python source distribution.⁴ A Python source distribution is much like a CPAN Perl distribution,⁵ except that it typically does not include dependency-checking information. This means that you will have to manage dependencies on your own.



Tip

Most Linux distributions provide an automatic or semi-automatic way to install packages like the ones Cedar Backup requires (think RPMs for Mandrake or RedHat, or Gentoo's Portage system). If you are not sure how to install these packages on your system, you might want to check out Appendix B, *Dependencies*. This appendix provides links to “upstream” source packages, plus as much information as I have been able to gather about packages for non-Debian platforms.

Installing Dependencies

Cedar Backup requires a number of external packages in order to function properly. Before installing Cedar Backup, you must make sure that these dependencies are met.

Cedar Backup is written in Python and requires version 2.3 or greater of the language. Version 2.3 was released on 29 July 2003, so by now most current Linux distributions should include it. Cedar Backup also requires one non-standard Python module, called PyXML (version 0.8.2 or better). You must install these Python dependencies on every peer node in a pool (master or client).

Additionally, remote client peer nodes must be running an *RSH-compatible* server, such as the **ssh** server.

Master machines require several other system utilities, most having to do with writing and validating CD media. On master machines, you must make sure that these utilities are available:

- **mkisofs**
- **cdrecord**
- **eject**

⁴See <http://docs.python.org/lib/module-distutils.html>.

⁵See <http://cpan.org/>.

- **mount**
- **unmount**
- **volname**
- An RSH-compatible client, such as **ssh**

Installing the Source Package

Python source packages are fairly easy to install. They are distributed as `.tar.gz` files which contain Python source code, a manifest and an installation script called `setup.py`.

Once you have downloaded the source package from the Cedar Solutions website,³ untar it:

```
$ zcat CedarBackup2-2.0.0.tar.gz | tar xvf -
```

This will create a directory called (in this case) `CedarBackup2-2.0.0`. The version number in the directory will always match the version number in the filename.

If you have root access and want to install the package to the “standard” Python location on your system, then you can install the package in two simple steps:

```
$ cd CedarBackup2-2.0.0
$ python setup.py install
```

Make sure that you are using Python 2.3 or better to execute `setup.py`.

Some users might want to choose a different install location or change other install parameters. To get more information about how `setup.py` works, use the `--help` option:

```
$ python setup.py --help
$ python setup.py install --help
```

In any case, once the package has been installed, you can proceed to configuration as described in Chapter 4, *Configuration*.

Chapter 4. Configuration

Overview

Configuring Cedar Backup is unfortunately somewhat complicated. The good news is that once you get through the initial configuration process, you'll hardly ever have to change anything. Even better, the most typical changes (i.e. adding and removing directories from a backup) are easy.

First, familiarize yourself with the concepts in Chapter 2, *Basic Concepts*. In particular, be sure that you understand the differences between a master and a client. (If you only have one machine, then your machine will act as both a master and a client, and we'll refer to your setup as a *pool of one*.) Then, install Cedar Backup per the instructions in Chapter 3, *Installation*.

Once everything has been installed, you are ready to begin configuring Cedar Backup. Look over the section called “Command Line Interface” (in Chapter 4, *Configuration*) to become familiar with the command line interface. Then, look over the section called “Configuration File Format” (below) and create a configuration file for each peer in your backup pool. To start with, create a very simple configuration file, then expand it later. Decide now whether you will store the configuration file in the standard place (`/etc/cback.conf`) or in some other location.

After you have all of the configuration files in place, configure each of your machines, following the instructions in the appropriate section below (for master, client or pool of one). Since the master and client(s) must communicate over the network, you won't be able to fully configure the master without configuring each client and vice-versa. The instructions are clear on what needs to be done.

Which Linux Distribution?

Cedar Backup has been designed for use on all Linux systems. However, since it was developed on a Debian system, and because I am a Debian developer, the packaging is prettier and the setup is somewhat simpler on a Debian system than on a system where you install from source.

The configuration instructions below have been generalized so they should work well regardless of what distribution you are running (i.e. RedHat, Gentoo, etc.). If instructions vary for a particular distribution, you will find a note related to that distribution.

I am always open to adding more distribution-specific hints and notes, so write me if you find problems with these instructions.

Command Line Interface

Syntax

The Cedar Backup command-line interface is implemented in the **cback** script. The **cback** script has the following syntax:

```
Usage: cback [switches] action(s)
```

The following switches are accepted:

-h, --help	Display this usage/help listing
-V, --version	Display version information
-b, --verbose	Print verbose output as well as logging to disk
-q, --quiet	Run quietly (display no output to the screen)
-c, --config	Path to config file (default: /etc/cback.conf)
-f, --full	Perform a full backup, regardless of configuration
-l, --logfile	Path to logfile (default: /var/log/cback.log)
-o, --owner	Logfile ownership, user:group (default: root:adm)
-m, --mode	Octal logfile permissions mode (default: 640)
-O, --output	Record some sub-command (i.e. tar) output to the log
-d, --debug	Write debugging information to the log (implies --output)
-s, --stack	Dump a Python stack trace instead of swallowing exceptions

The following actions may be specified:

all	Take all normal actions (collect, stage, store, purge)
collect	Take the collect action
stage	Take the stage action
store	Take the store action
purge	Take the purge action
rebuild	Rebuild "this week's" disc if possible
validate	Validate configuration only

You may also specify extended actions that have been defined in configuration.

You must specify at least one action to take. More than one of the "collect", "stage", "store" or "purge" actions and/or extended actions may be specified in any arbitrary order; they will be executed in a sensible order. The "all", "rebuild" or "validate" actions may not be combined with other actions.

Switches

-h, --help	Display usage/help listing.
-V, --version	Display version information.
-b, --verbose	Print verbose output to the screen as well writing to the logfile. When this option is enabled, most information that would normally be written to the logfile will also be written to the screen.
-q, --quiet	Run quietly (display no output to the screen).
-c, --config	Specify the path to an alternate configuration file. The default configuration file is /etc/cback.conf.
-f, --full	

Perform a full backup, regardless of configuration. For the collect action, this means that any existing information related to incremental backups will be ignored and rewritten; for the store action, this means that a new disc will be started.

-l, --logfile

Specify the path to an alternate logfile. The default logfile file is `/var/log/cback.log`.

-o, --owner

Specify the ownership of the logfile, in the form `user:group`. The default ownership is `root:adm`, to match the Debian standard for most logfiles. This value will only be used when creating a new logfile. If the logfile already exists when the **cback** script is executed, it will retain its existing ownership and mode. Only user and group names may be used, not numeric uid and gid values.

-m, --mode

Specify the permissions for the logfile, using the numeric mode as in `chmod(1)`. The default mode is `0640 (-rw-r----`). This value will only be used when creating a new logfile. If the logfile already exists when the **cback** script is executed, it will retain its existing ownership and mode.

-O, --output

Record some sub-command output to the logfile. When this option is enabled, all output from system commands will be logged. This might be useful for debugging or just for reference. Cedar Backup uses system commands mostly for dealing with the CD recorder and its media.

-d, --debug

Write debugging information to the logfile. This option produces a high volume of output, and would generally only be needed when debugging a problem. This option implies the `--output` option, as well.

-s, --stack

Dump a Python stack trace instead of swallowing exceptions. This forces Cedar Backup to dump the entire Python stack trace associated with an error, rather than just propagating last message it received back up to the user interface. Under some circumstances, this is useful information to include along with a bug report.

Actions

You can find more information about the various actions in the section called “The Backup Process” (in Chapter 2, *Basic Concepts*). In general, you may specify any combination of the `collect`, `stage`, `store` or `purge` actions, and the specified actions will be executed in a sensible order. Or, you can specify one of the `all`, `rebuild` or `validate` actions (but these actions may not be combined with other actions).

If you have configured any Cedar Backup extensions, then the actions associated with those extensions may also be specified on the command line. If you specify any other actions along with an extended action, the actions will be executed in a sensible order per configuration. The `all` action never executes extended actions, however.

Configuration File Format

Cedar Backup is configured through an XML ¹ configuration file, usually called `/etc/cback.conf`.

¹See <http://www.xml.com/pub/a/98/10/guide0.html> for a basic introduction to XML.

The configuration file contains the following sections: *reference*, *options*, *collect*, *stage*, *store*, *purge* and *extensions*.

All configuration files must contain the two general configuration sections, the reference section and the options section. Besides that, administrators need only configure actions they intend to use. For instance, on a client machine, administrators will generally only configure the collect and purge sections, while on a master machine they will have to configure all four action-related sections.² The extensions section is always optional and can be omitted unless extensions are in use.

Sample Configuration File

Both the Python source distribution and the Debian package come with a sample configuration file. The Debian package includes a stripped config file in `/etc/cback.conf` and a larger sample in `/usr/share/doc/cedar-backup2/cback.conf.sample`.

This is a sample configuration file similar to the one provided in the source package. Documentation below provides more information about each of the individual configuration sections.

```
<?xml version="1.0"?>
<cb_config>
  <reference>
    <author>Kenneth J. Pronovici</author>
    <revision>1.3</revision>
    <description>Sample</description>
  </reference>
  <options>
    <starting_day>tuesday</starting_day>
    <working_dir>/opt/backup/tmp</working_dir>
    <backup_user>backup</backup_user>
    <backup_group>group</backup_group>
    <rcp_command>/usr/bin/scp -B</rcp_command>
  </options>
  <collect>
    <collect_dir>/opt/backup/collect</collect_dir>
    <collect_mode>daily</collect_mode>
    <archive_mode>targz</archive_mode>
    <ignore_file>.cbignore</ignore_file>
    <dir>
      <abs_path>/etc</abs_path>
      <collect_mode>incr</collect_mode>
    </dir>
  </collect>
  <stage>
    <staging_dir>/opt/backup/staging</staging_dir>
    <peer>
      <name>debian</name>
      <type>local</type>
      <collect_dir>/opt/backup/collect</collect_dir>
    </peer>
  </stage>
  <store>
    <source_dir>/opt/backup/staging</source_dir>
    <media_type>cdrw-74</media_type>
    <device_type>cdwriter</device_type>
    <target_device>/dev/cdrw</target_device>
```

²See the section called “The Backup Process”, in Chapter 2, *Basic Concepts*.

```

        <target_scsi_id>0,0,0</target_scsi_id>
        <drive_speed>4</drive_speed>
        <check_data>Y</check_data>
    </store>
    <purge>
        <dir>
            <abs_path>/opt/backup/stage</abs_path>
            <retain_days>7</retain_days>
        </dir>
        <dir>
            <abs_path>/opt/backup/collect</abs_path>
            <retain_days>0</retain_days>
        </dir>
    </purge>
</cb_config>

```

Reference Configuration

The reference configuration section contains free-text elements that exist only for reference.. The section itself is required, but the individual elements may be left blank if desired.

This is an example reference configuration section:

```

<reference>
    <author>Kenneth J. Pronovici</author>
    <revision>Revision 1.3</revision>
    <description>Sample</description>
    <generator>Yet to be Written Config Tool (tm)</description>
</reference>

```

The following elements are part of the reference configuration section:

author

Author of the configuration file.

Restrictions: None

revision

Revision of the configuration file.

Restrictions: None

description

Description of the configuration file.

Restrictions: None

generator

Tool that generated the configuration file, if any.

Restrictions: None

Options Configuration

The options configuration section contains configuration options that are not specific to any one action. All of the fields are required and must not be empty.

This is an example options configuration section:

```
<options>
  <starting_day>tuesday</starting_day>
  <working_dir>/opt/backup/tmp</working_dir>
  <backup_user>backup</backup_user>
  <backup_group>backup</backup_group>
  <rcp_command>/usr/bin/scp -B</rcp_command>
</options>
```

The following elements are part of the options configuration section:

starting_day

Day that starts the week.

Cedar Backup is built around the idea of weekly backups. The starting day of week is the day that media will be rebuilt from scratch and that incremental backup information will be cleared.

Restrictions: Must be a day of the week in English, i.e. monday, tuesday, etc. The validation is case-sensitive.

working_dir

Working (temporary) directory to use for backups.

This directory is used for writing temporary files, such as tar file or ISO CD images as they are being built. It is also used to store day-to-day information about incremental backups.

The working directory should contain enough free space to hold temporary tar files (on a client) or to build an ISO CD image (on a master).

Restrictions: Must be an absolute path

backup_user

Effective user that backups should run as.

This user must exist on the machine which is being configured and should not be root (although that restriction is not enforced).

This value is also used as the default remote backup user for remote peers in the staging section.

Restrictions: Must be non-empty

backup_group

Effective group that backups should run as.

This group must exist on the machine which is being configured, and should not be root or some other “powerful” group (although that restriction is not enforced).

Restrictions: Must be non-empty

`rcp_command`

Default rcp-compatible copy command for staging.

The rcp command should be the exact command used for remote copies, including any required options. If you are using **scp**, you should pass it the **-B** option, so **scp** will not ask for any user input (which could hang the backup). A common example is something like **/usr/bin/scp -B**.

This value is used as the default value for all remote peers in the staging section. Technically, this value is not needed by clients, but we require it for all config files anyway.

Restrictions: Must be non-empty

Collect Configuration

The collect configuration section contains configuration options related the the collect action. This section contains a variable number of elements, including an optional exclusion section and a repeating subsection used to specify which directories to collect.

This is an example collect configuration section:

```
<collect>
  <collect_dir>/opt/backup/collect</collect_dir>
  <collect_mode>daily</collect_mode>
  <archive_mode>targz</archive_mode>
  <ignore_file>.cbignore</ignore_file>
  <exclude>
    <abs_path>/etc</abs_path>
    <pattern>.*\*.conf</pattern>
  </exclude>
  <dir>
    <abs_path>/etc</abs_path>
  </dir>
  <dir>
    <abs_path>/var/log</abs_path>
    <collect_mode>incr</collect_mode>
  </dir>
  <dir>
    <abs_path>/opt</abs_path>
    <collect_mode>weekly</collect_mode>
    <exclude>
      <abs_path>/opt/large</abs_path>
      <rel_path>backup</rel_path>
      <pattern>.*tmp</pattern>
    </exclude>
  </dir>
</collect>
```

The following elements are part of the collect configuration section:

`collect_dir`

Directory to collect files into.

On a client, this is the directory which tarfiles for individual collect directories are written into. The master then stages files from this directory into its own staging directory.

This field is always required. It must contain enough free space to collect all of the backed-up files on the machine in a compressed form.

Restrictions: Must be an absolute path

`collect_mode`

Default collect mode.

The collect mode describes how frequently a directory is backed up. See the section called “The Collect Action” (in Chapter 2, *Basic Concepts*) for more information.

This value is the collect mode that will be used by default during the collect process. Individual collect directories (below) may override this value. If *all* individual directories provide their own value, then this default value may be omitted from configuration.

Note: if your backup device does not support multisession discs, then you should probably use the `daily` collect mode to avoid losing data.

Restrictions: Must be one of `daily`, `weekly` or `incr`.

`archive_mode`

Default archive mode for collect files.

The archive mode maps to the way that a backup file is stored. A value `tar` means just a tarfile (`file.tar`); a value `targz` means a gzipped tarfile (`file.tar.gz`); and a value `tarbz2` means a bzipped tarfile (`file.tar.bz2`)

This value is the archive mode that will be used by default during the collect process. Individual collect directories (below) may override this value. If *all* individual directories provide their own value, then this default value may be omitted from configuration.

Restrictions: Must be one of `tar`, `targz` or `tarbz2`.

`ignore_file`

Default ignore file name.

The ignore file is an indicator file. If it exists in a given directory, then that directory will be recursively excluded from the backup as if it were explicitly excluded in configuration.

The ignore file provides a way for individual users (who might not have access to Cedar backup configuration) to control which of their own directories get backed up. For instance, users with a `~/tmp` directory might not want it backed up. If they create an ignore file in their directory (e.g. `~/tmp/.cbignore`), then Cedar Backup will ignore it.

This value is the ignore file name that will be used by default during the collect process. Individual collect directories (below) may override this value. If *all* individual directories provide their own value, then this default value may be omitted from configuration.

Restrictions: Must be non-empty

`exclude`

List of paths or patterns to exclude from the backup.

This is a subsection which contains a set of absolute paths and patterns to be excluded across all configured directories. For a given directory, the set of absolute paths and patterns to exclude is built from this list and any list that exists on the directory itself. Directories *cannot* override or remove entries that are in this list, however.

This section is optional, and if it exists can also be empty.

The exclude subsection can contain one or more of each of the following fields:

`abs_path`

An absolute path to be recursively excluded from the backup.

If a directory is excluded, then all of its children are also recursively excluded. For instance, a value `/var/log/apache` would exclude any files within `/var/log/apache` as well as files within other directories under `/var/log/apache`.

This field can be repeated as many times as is necessary.

Restrictions: Must be an absolute path.

`pattern`

A pattern to be recursively excluded from the backup.

The pattern must be a Python regular expression.³ It is assumed to be bounded at front and back by the beginning and end of the string (i.e. it is treated as if it begins with `^` and ends with `$`).

If the pattern causes a directory to be excluded, then all of the children of that directory are also recursively excluded. For instance, a value `.*apache.*` might match the `/var/log/apache` directory. This would exclude any files within `/var/log/apache` as well as files within other directories under `/var/log/apache`.

This field can be repeated as many times as is necessary.

Restrictions: Must be non-empty

`dir`

A directory to be collected.

This is a subsection which contains information about a specific directory to be collected (backed up).

This section can be repeated as many times as is necessary. At least one collect directory must be configured.

The collect directory subsection contains the following fields:

`abs_path`

Absolute path of the directory to collect.

The path may be either a directory, a soft link to a directory, or a hard link to a directory. All three are treated the same at this level.

³See <http://docs.python.org/lib/re-syntax.html>

The contents of the directory will be recursively collected. The backup will contain all of the files in the directory, as well as the contents of all of the subdirectories within the directory, etc.

Soft links *within* the directory are treated as files, i.e. they are copied verbatim (as a link) and their contents are not backed up.

Restrictions: Must be an absolute path.

`collect_mode`
Collect mode for this directory

The collect mode describes how frequently a directory is backed up. See the section called “The Collect Action” (in Chapter 2, *Basic Concepts*) for more information.

This field is optional. If it doesn't exist, the backup will use the default collect mode.

Note: if your backup device does not support multisession discs, then you should probably confine yourself to the `daily` collect mode, to avoid losing data.

Restrictions: Must be one of `daily`, `weekly` or `incr`.

`archive_mode`
Archive mode for this directory.

The archive mode maps to the way that a backup file is stored. A value `tar` means just a tarfile (`file.tar`); a value `targz` means a gzipped tarfile (`file.tar.gz`); and a value `tarbz2` means a bzipped tarfile (`file.tar.bz2`)

This field is optional. if it doesn't exist, the backup will use the default archive mode.

Restrictions: Must be one of `tar`, `targz` or `tarbz2`.

`ignore_file`
Ignore file name for this directory.

The ignore file is an indicator file. If it exists in a given directory, then that directory will be recursively excluded from the backup as if it were explicitly excluded in configuration.

The ignore file provides a way for individual users (who might not have access to Cedar backup configuration) to control which of their own directories get backed up. For instance, users with a `~/tmp` directory might not want it backed up. If they create an ignore file in their directory (e.g. `~/tmp/.cbignore`), then Cedar Backup will ignore it.

This field is optional. If it doesn't exist, the backup will use the default ignore file name.

Restrictions: Must be non-empty

`exclude`
List of paths or patterns to exclude from the backup.

This is a subsection which contains a set of paths and patterns to be excluded within this collect directory. This list is combined with the program-wide list to build a complete list for the directory.

This section is entirely optional, and if it exists can also be empty.

The exclude subsection can contain one or more of each of the following fields:

`abs_path`

An absolute path to be recursively excluded from the backup.

If a directory is excluded, then all of its children are also recursively excluded. For instance, a value `/var/log/apache` would exclude any files within `/var/log/apache` as well as files within other directories under `/var/log/apache`.

This field can be repeated as many times as is necessary.

Restrictions: Must be an absolute path.

`rel_path`

A relative path to be recursively excluded from the backup.

The path is assumed to be relative to the collect directory itself. For instance, if the configured directory is `/opt/web` a configured relative path of `something/else` would exclude the path `/opt/web/something/else`.

If a directory is excluded, then all of its children are also recursively excluded. For instance, a value `something/else` would exclude any files within `something/else` as well as files within other directories under `something/else`.

This field can be repeated as many times as is necessary.

Restrictions: Must be non-empty.

`pattern`

A pattern to be excluded from the backup.

The pattern must be a Python regular expression.³ It is assumed to be bounded at front and back by the beginning and end of the string (i.e. it is treated as if it begins with `^` and ends with `$`).

If the pattern causes a directory to be excluded, then all of the children of that directory are also recursively excluded. For instance, a value `*apache.*` might match the `/var/log/apache` directory. This would exclude any files within `/var/log/apache` as well as files within other directories under `/var/log/apache`.

This field can be repeated as many times as is necessary.

Restrictions: Must be non-empty

Stage Configuration

The stage configuration section contains configuration options related the the stage action. The section defines the set of peers in a backup pool, and then also indicates where data from those peers should be staged to.

This is an example stage configuration section:

```
<stage>
  <staging_dir>/opt/backup/stage</staging_dir>
```

```
<peer>
  <name>machine1</name>
  <type>local</type>
  <collect_dir>/opt/backup/collect</collect_dir>
</peer>
<peer>
  <name>machine2</name>
  <type>remote</type>
  <backup_user>backup</backup_user>
  <collect_dir>/opt/backup/collect</collect_dir>
</peer>
</stage>
```

The following elements are part of the stage configuration section:

staging_dir

Directory to stage files into.

This is the directory into which the master stages collected data from each of the clients. Within the staging directory, data is staged into date-based directories by peer name. For instance, peer “daystrom” backed up on 19 Feb 2005 would be staged into something like 2005/02/19/daystrom relative to the staging directory itself.

This field is always required. The directory must contain enough free space to stage all of the files collected from all of the various machines in a backup pool. Many administrators set up purging to keep staging directories around for a week or more, which require even more space.

Restrictions: Must be an absolute path

peer (local version)

Local client peer in a backup pool.

This is a subsection which contains information about a specific local client peer to be staged (backed up). A local peer is one whose collect directory can be reached without requiring any rsh-based network calls. It is possible that a remote peer might be staged as a local peer if its collect directory is mounted to the master via NFS, AFS or some other method.

This section can be repeated as many times as is necessary. At least one remote or local peer must be configured.

The local peer subsection must contain the following fields:

name

Name of the peer, typically a valid hostname.

For local peers, this value is only used for reference. However, it is good practice to list the peer's hostname here, for consistency with remote peers.

Restrictions: Must be non-empty.

type

Type of this peer.

This value identifies the type of the peer. For a local peer, it must always be `local`.

Restrictions: Must be `local`.

`collect_dir`

Collect directory to stage from for this peer.

The master will copy all files in this directory into the appropriate staging directory. Since this is a local peer, the directory is assumed to be reachable via normal filesystem operations (i.e. **cp**).

Restrictions: Must be an absolute path.

`peer` (remote version)

Remote client peer in a backup pool.

This is a subsection which contains information about a specific remote client peer to be staged (backed up). A remote peer is one whose collect directory can only be reached via an rsh-based network call.

This section can be repeated as many times as is necessary. At least one remote or local peer must be configured.

The remote peer subsection must contain the following fields:

`name`

Hostname of the peer.

For remote peers, this must be a valid DNS hostname or IP address which can be resolved during an rsh-based network call.

Restrictions: Must be non-empty.

`type`

Type of this peer.

This value identifies the type of the peer. For a remote peer, it must always be `remote`.

Restrictions: Must be `remote`.

`collect_dir`

Collect directory to stage from for this peer.

The master will copy all files in this directory into the appropriate staging directory. Since this is a remote peer, the directory is assumed to be reachable via rsh-based network operations (i.e. **scp** or the configured `rcp` command).

Restrictions: Must be an absolute path.

`backup_user`

Name of backup user on the remote peer.

This username will be used when copying files from the remote peer via an rsh-based network connection.

This field is optional. if it doesn't exist, the backup will use the default backup user from the options

section.

Restrictions: Must be non-empty.

`rcp_command`

The rcp-compatible copy command for this peer.

The rcp command should be the exact command used for remote copies, including any required options. If you are using **scp**, you should pass it the **-B** option, so **scp** will not ask for any user input (which could hang the backup). A common example is something like **/usr/bin/scp -B**.

This field is optional. if it doesn't exist, the backup will use the default rcp command from the options section.

Restrictions: Must be non-empty.

Store Configuration

The store configuration section contains configuration options related the the store action. This section contains several optional fields. Most fields control the way media is written using the writer device.

This is an example store configuration section:

```
<store>
  <source_dir>/opt/backup/stage</source_dir>
  <media_type>cdrw-74</media_type>
  <device_type>cdwriter</device_type>
  <target_device>/dev/cdrw</target_device>
  <target_scsi_id>0,0,0</target_scsi_id>
  <drive_speed>4</drive_speed>
  <check_data>Y</check_data>
</store>
```

The following elements are part of the store configuration section:

`source_dir`

Directory whose contents should be written to media.

This directory *must* be a Cedar Backup staging directory, as configured in the staging configuration section. Only certain data from that directory (typically, data from the current day) will be written to disc.

Restrictions: Must be an absolute path

`media_type`

Type of the media in the device.

Unless you want to throw away a backup disc every week, you are probably best off using rewritable media.

If you have no idea what kind of media you have, choose `cdr-74`. For more information on media

types, see the section called “Media and Device Types” (in Chapter 2, *Basic Concepts*).

Restrictions: Must be one of `cdr-74`, `cdrw-74`, `cdr-80` or `cdrw-80`.

`device_type`

Type of the device used to write the media.

This field mostly exists for planned future enhancements, such as support for DVD writers. It indicates what type of device should be used to write the media, in case that makes a difference to the underlying writer functionality. Currently, it can only be set to `cdwriter`.

This field is optional. If it doesn't exist, the `cdwriter` device type is assumed.

Restrictions: If set, must be `cdwriter`.

`target_device`

Filesystem device name for writer device.

This is the UNIX device name for the writer drive, for instance `/dev/scd0` or `/dev/cdrw`. The device name is not needed in order to write to the media. However, it is needed in order to do several pre-write checks (such as whether the device might already be mounted) as well as the post-write consistency check, if enabled.

Restrictions: Must be an absolute path.

`target_scsi_id`

SCSI id for writer device

Under Linux, CD burners generally only work through a SCSI or pseudo-SCSI interface. This value is the SCSI id for the drive in the form `scsibus,target,lun`. The `ATA:` or `ATAPI:` prefixes are also allowed.^{4 5}

Restrictions: Must be a valid SCSI identifier.

`drive_speed`

Speed of the drive, i.e. 2 for a 2x device.

This field is optional. If it doesn't exist, the underlying device-related functionality will use the default drive speed. Since some media is speed-sensitive, it might be a good idea to set this to a sensible value for your writer.

Restrictions: If set, must be an integer ≥ 1 .

`check_data`

Indicates whether the media should be validated.

This field indicates whether a resulting image on the media should be validated after the write completes, by running a consistency check against it. If this check is enabled, the contents of the staging directory are directly compared to the media, and an error is reported if there is a mismatch.

Practice shows that some drives can encounter an error when writing a multisession disc, but not report any problems. This consistency check allows us to catch the problem. By default, the consistency check is disabled, but most users should choose to enable it unless they have a good

⁴For more information on how to configure your CD-R/CD-RW device, see <http://www.tldp.org/HOWTO/CDROM-HOWTO>

⁵See <http://www.tldp.org/HOWTO/ATA-RAID-HOWTO/index.html>

reason not to.

This field is optional. If it doesn't exist, then N will be assumed.

Restrictions: Must be a boolean (Y or N).

Purge Configuration

The purge configuration section contains configuration options related the the purge action. This section contains a set of directories to be purged, along with information about the schedule at which they should be purged.

Typically, Cedar Backup should be configured to purge collect directories daily (retain days of 0).

If you are tight on space, staging directories can also be purged daily. However, if you have space to spare, you should consider purging about once per week. That way, if your backup media is damaged, you will be able to recreate the week's backup using the rebuild action.

You should also purge the working directory periodically, once every few weeks or once per month. This way, if any unneeded files are left around, perhaps because a backup was interrupted or because configuration changed, they will eventually be removed. *The working directory should not be purged any more frequently than once per week, otherwise you will risk destroying data used for incremental backups.*

This is an example purge configuration section:

```
<purge>
  <dir>
    <abs_path>/opt/backup/stage</abs_path>
    <retain_days>7</retain_days>
  </dir>
  <dir>
    <abs_path>/opt/backup/collect</abs_path>
    <retain_days>0</retain_days>
  </dir>
</purge>
```

The following elements are part of the purge configuration section:

dir

A directory to purge within.

This is a subsection which contains information about a specific directory to purge within.

This section can be repeated as many times as is necessary. At least one purge directory must be configured.

The purge directory subsection contains the following fields:

abs_path

Absolute path of the directory to purge within.

The contents of the directory will be purged based on age. The purge will remove any files that were last modified more than “retain days” days ago. Empty directories will also eventually be removed. The purge directory itself will never be removed.

The path may be either a directory, a soft link to a directory, or a hard link to a directory. Soft links *within* the directory (if any) are treated as files.

Restrictions: Must be an absolute path.

`retain_days`
Number of days to retain old files.

Once it has been more than this many days since a file was last modified, it is a candidate for removal.

Restrictions: Must be an integer ≥ 0 .

Extensions Configuration

The extensions configuration section is used to configure third-party extensions to Cedar Backup. If you don't intend to use any extensions, or don't know what extensions are, then you can safely leave this section out of your configuration file. It is optional.

Extensions configuration is used to specify “extended actions” implemented by code external to Cedar Backup. An administrator can use this section to map command-line Cedar Backup actions to third-party extension functions.

Each extended action has a name, which is mapped to a Python function within a particular module. Each action also has an index associated with it. This index is used to properly order execution when more than one action is specified on the command line. The standard actions have predefined indexes, and extended actions are interleaved into the normal order of execution using those indexes. The collect action has index 100, the stage index has action 200, the store action has index 300 and the purge action has index 400.

For instance, imagine that a third-party developer provided a Cedar Backup extension to back up a certain kind of database repository, and you wanted to map that extension to the “database” command-line action. You have been told that this function is called “foobar.database”. You think of this backup as a “collect” kind of action, so you want it to be performed after collect but before stage and purge if more than one action is specified on the command line.

To configure this extension, you would list an action with a name “database”, a module “foobar”, a function name “database” and an index of “101”.

This is how the hypothetical action would be configured:

```
<extensions>
  <action>
    <name>database</name>
    <module>foobar</module>
    <function>database</function>
    <index>101</index>
  </action>
</extensions>
```

The following elements are part of the extensions configuration section:

action

This is a subsection that contains configuration related to a single extended action.

This section can be repeated as many times as is necessary.

The action subsection contains the following fields:

name

Name of the extended action.

Restrictions: Must be a non-empty string consisting of only lower-case letters and digits.

module

Name of the Python module associated with the extension function.

Restrictions: Must be a non-empty string and a valid Python identifier.

function

Name of the Python extension function within the module.

Restrictions: Must be a non-empty string and a valid Python identifier.

index

Index of action, for execution ordering.

Restrictions: Must be an integer ≥ 0 .

Setting up a Pool of One

Cedar Backup has been designed primarily for situations where there is a single master and a set of other clients that the master interacts with. However, it will just as easily work for a single machine (a backup pool of one).

Once you complete all of these configuration steps, your backups will run as scheduled out of cron. Any errors that occur will be reported in daily emails to your root user (or the user that receives root's email). If you don't receive any emails, then you know your backup worked.

Note: all of these configuration steps should be run as the root user, unless otherwise indicated.

Step 1: Make sure email works.

Cedar Backup relies on email for problem notification. This notification works through the magic of cron. Cron will email any output from each job it executes to the user associated with the job. Since by default Cedar Backup only writes output to the terminal if errors occur, this ensures that notification emails will only be sent out if errors occur.

In order to receive problem notifications, you must make sure that email works for the user which is

running the Cedar Backup cron jobs (typically root). Refer to your distribution's documentation for information on how to configure email on your system. Note that you may prefer to configure root's email to forward to some other user, so you do not need to check the root user's mail in order to see Cedar Backup errors.

Step 2: Configure your CD-R or CD-RW drive.

Your CD-R or CD-RW drive must either be a SCSI device or must be configured to act like a Linux SCSI device.⁴ Regardless of what kind of drive you have, make sure you know its SCSI address (in the form `scsibus, target, lun`) and its device name (i.e. `/dev/cdrw`). The SCSI address will be used to write to media, and the device name will be used when Cedar Backup needs to mount the media (for instance, when a validation check must be run).

If you have an IDE drive rather than a SCSI drive and are using the IDE-to-SCSI interface (which is the norm for most IDE drives under Linux), then be prepared to enter the simulated SCSI device address, which is often `0, 0, 0`. Newer Linux kernels (2.6.x) can also be compiled with support for other kinds of CD drive interfaces. If you have configured your CD drive to use *ATA*⁵ or *ATAPI*,⁶ then include this prefix in your simulated device address, i.e. `ATA: 0, 0, 0` or `ATAPI: 0, 0, 0`.

Step 3: Configure your backup user.

Choose a user to be used for backups. Some Linux distributions may come with a “ready made” backup user. For other distributions, you may have to create a user yourself. You may choose any id you like, but a descriptive name such as `backup` or `cback` is a good choice. See your distribution's documentation for information on how to add a user.



Note

Standard Debian systems come with a user named `backup`. You may choose to stay with this user or create another one.

Step 4: Create your backup tree.

Cedar Backup requires a backup directory tree on disk. This directory tree must be roughly three times as big as the amount of data that will be backed up on a nightly basis, to allow for the data to be collected, staged, and then placed into an ISO CD image on disk. (This is one disadvantage to using Cedar Backup in single-machine pools, but in this day of really large hard drives, it might not be an issue.) Note that if you elect not to purge the staging directory every night, you will need even more space.

You should create a collect directory, a staging directory and a working (temporary) directory. One recommended layout is this:

```
/opt/  
  backup/  
    collect/  
    stage/  
    tmp/
```

⁶ As of kernel 2.6, you can use ATAPI directly, without SCSI emulation, by prepending `ATAPI:` to the device address.

If you will be backing up sensitive information (i.e. password files), it is recommended that these directories be owned by the backup user (whatever you named it), with permissions 700.



Note

You don't have to use `/opt` as the root of your directory structure. Use anything you would like. I use `/opt` because it is my “dumping ground” for filesystems that Debian does not manage.

Some users have requested that the Debian packages set up a more “standard” location for backups right out-of-the-box. I have resisted doing this because it's difficult to choose an appropriate backup location from within the package. If you would prefer, you can create the backup directory structure within some existing Debian directory such as `/var/backups` or `/var/tmp`.

Step 5: Modify the backup cron jobs.

There are four parts to a Cedar Backup run: collect, stage, store and purge. The usual way of setting off these steps is through a cron job. For more information on using cron, see the manpage for `crontab(5)`.

Backing up large directories and creating ISO CD images can be intensive operations, and could slow your computer down significantly. Choose a backup time that will not interfere with normal use of your computer. Usually, you will want the backup to occur every day, but it is possible to configure cron to execute the backup only one day per week, three days per week, etc.

Since Cedar Backup should be run as root, one way to configure the cron job is to add a line like this to your `/etc/crontab` file:

```
30 00 * * * root cback all
```

Or, you can create an executable script containing just these lines and place that file in the `/etc/cron.daily` directory:

```
#!/bin/sh
cback all
```

You should consider adding the `--output` or `-O` switch to your **cback** command-line in cron. This will result in larger logs, but could help diagnose problems when commands like **cdrecord** or **mkisofs** fail mysteriously.



Note

On a Debian system, execution of daily backups is controlled by the file `/etc/cron.d/cedar-backup2`. As installed, this file contains several different settings, all commented out. Uncomment the “Single machine (pool of one)” entry in the file, and change the line so that the backup goes off when you want it to.

Step 6: Create the Cedar Backup configuration file.

Following the instructions in the section called “Configuration File Format” (above) create a configuration file for your machine. Since you are working with a pool of one, you must configure all four action-specific sections: collect, stage, store and purge.

The usual location for the Cedar Backup config file is `/etc/cback.conf`. If you change the location, make sure you edit your cronjobs (step 5) to point the **cback** script at the correct config file (using the `--config` option).



Warning

Configuration files should always be writable only by root (or by the file owner, if the owner is not root).

If you intend to place confidential information into the Cedar Backup configuration file, make sure that you set the filesystem permissions on the file appropriately. For instance, if you configure any extensions that require passwords or other similar information, you should make the file readable only to root or to the file owner, if the owner is not root.

Step 7: Validate the Cedar Backup configuration file.

Use the command **cback validate** to validate your configuration file. This command checks that the configuration file can be found and parsed, and also checks for typical configuration problems, such as invalid CD-R/CD-RW device entries.

Note: the most common cause of configuration problems is in not closing XML tags properly. Any XML tag that is “opened” must be “closed” appropriately.

Step 8: Test your backup.

Place a valid CD-R or CD-RW disc in your drive, and then use the command **cback --full all**. You should execute this command as root. If the command completes with no output, then the backup was run successfully.

Just to be sure that everything worked properly, check the logfile (`/var/log/cback.log`) for errors and also mount the CD-R or CD-RW disc to be sure it can be read.

If Cedar Backup ever completes “normally” but the disc that is created is not usable, please report this as a bug.⁷ To be safe, always enable the consistency check option in the store configuration section.

Setting up a Client Peer Node

Cedar Backup has been designed to backup entire “pools” of machines. In any given pool, there is one master and some number of clients. Most of the work takes place on the master, so configuring a client is a little simpler than configuring a master.

⁷ See <http://cedar-solutions.com/bugzilla/>.

Backups are designed to take place over an RSH or SSH connection. Because RSH is generally considered insecure, you are encouraged to use SSH rather than RSH. This document will only describe how to configure Cedar Backup to use SSH; if you want to use RSH, you're on your own.

Once you complete all of these configuration steps, your backups will run as scheduled out of cron. Any errors that occur will be reported in daily emails to your root user (or the user that receives root's email). If you don't receive any emails, then you know your backup worked.

Note: all of these configuration steps should be run as the root user, unless otherwise indicated.

Step 1: Make sure email works.

Cedar Backup relies on email for problem notification. This notification works through the magic of cron. Cron will email any output from each job it executes to the user associated with the job. Since by default Cedar Backup only writes output to the terminal if errors occur, this neatly ensures that notification emails will only be sent out if errors occur.

In order to receive problem notifications, you must make sure that email works for the user which is running the Cedar Backup cron jobs (typically root). Refer to your distribution's documentation for information on how to configure email on your system. Note that you may prefer to configure root's email to forward to some other user, so you do not need to check the root user's mail in order to see Cedar Backup errors.

Step 2: Configure the master in your backup pool.

You will not be able to complete the client configuration until at least step 3 of the master's configuration has been completed. In particular, you will need to know the master's public SSH identity to fully configure a client.

To find the master's public SSH identity, log in as the backup user on the master and **cat** the public identity file `~/.ssh/id_rsa.pub`:

```
user@machine> cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEA0vOKj1fwohPg1oPRdrmwHk75l3mI9Tb/WRZfVnu2Pw69
uyphM9wBLRo6QfOC2T8vZCB8o/ZIgtAM3tkM0UgQHxKBXAZ+H36TOgg7BcI20I93iGtzpsMA/uXQy8kH
HgZooYqQ9pw+ZduXgmPcAAv2b5eTm07wRqFt/U84k6bhTzs= user@machine
```

Step 3: Configure your backup user.

Choose a user to be used for backups. Some Linux distributions may come with a "ready made" backup user. For other distributions, you may have to create a user yourself. You may choose any id you like, but a descriptive name such as `backup` or `cback` is a good choice. See your distribution's documentation for information on how to add a user.



Note

Standard Debian systems come with a user named `backup`. You may choose to stay with this user or create another one.

Once you have created your backup user, you must create an SSH keypair for it. Log in as your backup user, and then run the command **ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa**:

```
user@machine> ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/user/.ssh'.
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
11:3e:ad:72:95:fe:96:dc:1e:3b:f4:cc:2c:ff:15:9e user@machine
```

The default permissions for this directory should be fine. However, if the directory existed before you ran **ssh-keygen**, then you may need to modify the permissions. Make sure that the `~/.ssh` directory is readable only by the backup user (i.e. mode 700), that the `~/.ssh/id_rsa` file is only readable and writable only by the backup user (i.e. mode 600) and that the `~/.ssh/id_rsa.pub` file is writable only by the backup user (i.e. mode 600 or mode 644).

Finally, take the master's public SSH identity (which you found in step 2) and cut-and-paste it into the file `~/.ssh/authorized_keys`. Make sure the identity value is pasted into the file *all on one line*, and that the `authorized_keys` file is owned by your backup user and has permissions 600.

If you have other preferences or standard ways of setting up your users' SSH configuration (i.e. different key type, etc.), feel free to do things your way. The important part is that the master must be able to SSH into a client *with no password entry required*.

Step 4: Create your backup tree.

Cedar Backup requires a backup directory tree on disk. This directory tree must be roughly as big as the amount of data that will be backed up on a nightly basis (more if you elect not to purge it all every night).

You should create a collect directory and a working (temporary) directory. One recommended layout is this:

```
/opt/
  backup/
    collect/
    tmp/
```

If you will be backing up sensitive information (i.e. password files), it is recommended that these directories be owned by the backup user (whatever you named it), with permissions 700.



Note

You don't have to use `/opt` as the root of your directory structure. Use anything you would like. I use `/opt` because it is my “dumping ground” for filesystems that Debian does not manage.

Some users have requested that the Debian packages set up a more "standard" location for

backups right out-of-the-box. I have resisted doing this because it's difficult to choose an appropriate backup location from within the package. If you would prefer, you can create the backup directory structure within some existing Debian directory such as `/var/backups` or `/var/tmp`.

Step 5: Modify the backup cron jobs.

There are two parts to a Cedar Backup run on a client: collect and purge. The usual way of setting off these steps is through a cron job. For more information on using cron, see the manpage for `crontab(5)`.

Backing up large directories could slow your computer down significantly. Choose a backup time that will not interfere with normal use of your computer. Usually, you will want the backup to go occur every day, but it is possible to configure cron to execute the backup only one day per week, three days per week, etc.

Since Cedar Backup should be run as root, you should add a set of lines like this to your `/etc/crontab` file:

```
30 00 * * * root cback collect
30 06 * * * root cback purge
```

You should consider adding the `--output` or `-O` switch to your **cback** command-line in cron. This will result in larger logs, but could help diagnose problems when commands like **cdrecord** or **mkisofs** fail mysteriously.

You will need to coordinate the collect and purge actions on the client so that the collect action completes before the master attempts to stage, and so that the purge action does not begin until after the master has completed staging. Usually, allowing an hour or two between steps should be sufficient.⁸



Note

On a Debian system, execution of daily backups is controlled by the file `/etc/cron.d/cedar-backup2`. As installed, this file contains several different settings, all commented out. Uncomment the “Client machine” entries in the file, and change the lines so that the backup goes off when you want it to.

Step 6: Create the Cedar Backup configuration file.

Following the instructions in the section called “Configuration File Format” (above), create a configuration file for your machine. Since you are working with a client, you must configure all action-specific sections for the collect and purge actions.

The usual location for the Cedar Backup config file is `/etc/cback.conf`. If you change the location, make sure you edit your cronjobs (step 5) to point the **cback** script at the correct config file (using the `--config` option).

⁸See the section called “Coordination between Master and Clients” in Chapter 2, *Basic Concepts*.



Warning

Configuration files should always be writable only by root (or by the file owner, if the owner is not root).

If you intend to place confidential information into the Cedar Backup configuration file, make sure that you set the filesystem permissions on the file appropriately. For instance, if you configure any extensions that require passwords or other similar information, you should make the file readable only to root or to the file owner, if the owner is not root.

Step 7: Validate the Cedar Backup configuration file.

Use the command **cback validate** to validate your configuration file. This command checks that the configuration file can be found and parsed, and also checks for typical configuration problems. This command *only* validates configuration on the one client, not the master or any other clients in a pool.

Note: the most common cause of configuration problems is in not closing XML tags properly. Any XML tag that is “opened” must be “closed” appropriately.

Step 8: Test your backup.

Use the command **cback --full collect purge**. If the command completes with no output, then the backup was run successfully. Just to be sure that everything worked properly, check the logfile (`/var/log/cback.log`) for errors.

Setting up a Master Peer Node

Cedar Backup has been designed to backup entire “pools” of machines. In any given pool, there is one master and some number of clients. Most of the work takes place on the master, so configuring a master is somewhat more complicated than configuring a client.

Backups are designed to take place over an RSH or SSH connection. Because RSH is generally considered insecure, you are encouraged to use SSH rather than RSH. This document will only describe how to configure Cedar Backup to use SSH; if you want to use RSH, you're on your own.

Once you complete all of these configuration steps, your backups will run as scheduled out of cron. Any errors that occur will be reported in daily emails to your root user (or whichever other user receives root's email). If you don't receive any emails, then you know your backup worked.

Note: all of these configuration steps should be run as the root user, unless otherwise indicated.

Step 1: Make sure email works.

Cedar Backup relies on email for problem notification. This notification works through the magic of cron. Cron will email any output from each job it executes to the user associated with the job. Since by default Cedar Backup only writes output to the terminal if errors occur, this neatly ensures that notification emails will only be sent out if errors occur.

In order to receive problem notifications, you must make sure that email works for the user which is running the Cedar Backup cron jobs (typically root). Refer to your distribution's documentation for information on how to configure email on your system. Note that you may prefer to configure root's

email to forward to some other user, so you do not need to check the root user's mail in order to see Cedar Backup errors.

Step 2: Configure your CD-R or CD-RW drive.

Your CD-R or CD-RW drive must either be a SCSI device or must be configured to act like a Linux SCSI device.⁴ Regardless of what kind of drive you have, make sure you know its SCSI address (in the form `scsibus, target, lun`) and its device name (i.e. `/dev/cdrw`). The SCSI address will be used to write to media, and the device name will be used when Cedar Backup needs to mount the media (for instance, when a validation check must be run).

If you have an IDE drive rather than a SCSI drive and are using the IDE-to-SCSI interface (which is the norm for most IDE drives under Linux), then be prepared to enter the simulated SCSI device address, which is often `0, 0, 0`. Newer Linux kernels (2.6.x) can also be compiled with support for other kinds of CD drive interfaces. If you have configured your CD drive to use *ATA*⁵ or *ATAPI*,⁶ then include this prefix in your simulated device address, i.e. `ATA:0, 0, 0` or `ATAPI:0, 0, 0`.

Step 3: Configure your backup user.

Choose a user to be used for backups. Some Linux distributions may come with a “ready made” backup user. For other distributions, you may have to create a user yourself. You may choose any id you like, but a descriptive name such as `backup` or `cback` is a good choice. See your distribution's documentation for information on how to add a user.



Note

Standard Debian systems come with a user named `backup`. You may choose to stay with this user or create another one.

Once you have created your backup user, you must create an SSH keypair for it. Log in as your backup user, and then run the command `ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa`:

```
user@machine> ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/user/.ssh'.
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
11:3e:ad:72:95:fe:96:dc:1e:3b:f4:cc:2c:ff:15:9e user@machine
```

The default permissions for this directory should be fine. However, if the directory existed before you ran `ssh-keygen`, then you may need to modify the permissions. Make sure that the `~/.ssh` directory is readable only by the backup user (i.e. mode 700), that the `~/.ssh/id_rsa` file is only readable and writable by the backup user (i.e. mode 600) and that the `~/.ssh/id_rsa.pub` file is writable only by the backup user (i.e. mode 600 or mode 644).

If you have other preferences or standard ways of setting up your users' SSH configuration (i.e. different key type, etc.), feel free to do things your way. The important part is that the master must be able to SSH into a client *with no password entry required*.

Step 4: Create your backup tree.

Cedar Backup requires a backup directory tree on disk. This directory tree must be roughly large enough hold twice as much data as will be backed up from the entire pool on a given night, plus space for whatever is collected on the master itself. This will allow for all three operations - collect, stage and store - to have enough space to complete. Note that if you elect not to purge the staging directory every night, you will need even more space.

You should create a collect directory, a staging directory and a working (temporary) directory. One recommended layout is this:

```
/opt/  
  backup/  
    collect/  
    stage/  
    tmp/
```

If you will be backing up sensitive information (i.e. password files), it is recommended that these directories be owned by the backup user (whatever you named it), with permissions 700.



Note

You don't have to use `/opt` as the root of your directory structure. Use anything you would like. I use `/opt` because it is my “dumping ground” for filesystems that Debian does not manage.

Some users have requested that the Debian packages set up a more “standard” location for backups right out-of-the-box. I have resisted doing this because it's difficult to choose an appropriate backup location from within the package. If you would prefer, you can create the backup directory structure within some existing Debian directory such as `/var/backups` or `/var/tmp`.

Step 5: Modify the backup cron jobs.

There are four parts to a Cedar Backup run: collect, stage, store and purge. The usual way of setting off these steps is through a cron job. For more information on using cron, see the manpage for `crontab(5)`.

Backing up large directories and creating ISO CD images can be intensive operations, and could slow your computer down significantly. Choose a backup time that will not interfere with normal use of your computer. Usually, you will want the backup to go occur every day, but it is possible to configure cron to execute the backup only one day per week, three days per week, etc.

Since Cedar Backup should be run as root, you should add a set of lines like this to your `/etc/crontab` file:

```
30 00 * * * root    cback collect  
30 02 * * * root    cback stage  
30 04 * * * root    cback store  
30 06 * * * root    cback purge
```

You should consider adding the `--output` or `-O` switch to your **cback** command-line in cron. This will result in larger logs, but could help diagnose problems when commands like **cdrecord** or **mkisofs** fail mysteriously.

You will need to coordinate the collect and purge actions on clients so that their collect actions complete before the master attempts to stage, and so that their purge actions do not begin until after the master has completed staging. Usually, allowing an hour or two between steps should be sufficient.⁸



Note

On a Debian system, execution of daily backups is controlled by the file `/etc/cron.d/cedar-backup2`. As installed, this file contains several different settings, all commented out. Uncomment the “Master machine” entries in the file, and change the lines so that the backup goes off when you want it to.

Step 6: Create the Cedar Backup configuration file.

Following the instructions in the section called “Configuration File Format” (above), create a configuration file for your machine. Since you are working with a master machine, you must configure all four action-specific sections: collect, stage, store and purge.

The usual location for the Cedar Backup config file is `/etc/cback.conf`. If you change the location, make sure you edit your cronjobs (step 5) to point the **cback** script at the correct config file (using the `--config` option).



Warning

Configuration files should always be writable only by root (or by the file owner, if the owner is not root).

If you intend to place confidential information into the Cedar Backup configuration file, make sure that you set the filesystem permissions on the file appropriately. For instance, if you configure any extensions that require passwords or other similar information, you should make the file readable only to root or to the file owner, if the owner is not root.

Step 7: Validate the Cedar Backup configuration file.

Use the command **cback validate** to validate your configuration file. This command checks that the configuration file can be found and parsed, and also checks for typical configuration problems, such as invalid CD-R/CD-RW device entries. This command *only* validates configuration on the master, not any clients that the master might be configured to connect to.

Note: the most common cause of configuration problems is in not closing XML tags properly. Any XML tag that is “opened” must be “closed” appropriately.

Step 8: Test connectivity to client machines.

This step must wait until after your client machines have been at least partially configured. Once the backup user(s) have been configured on the client machine(s) in a pool, attempt an SSH connection to each client.

Log in as the backup user on the master, and then use the command **ssh user@machine** where *user* is the name of backup user *on the client machine*, and *machine* is the name of the client machine.

If you are able to log in successfully without entering a password, then things have been configured properly. Otherwise, double-check that you followed the user setup instructions for the master and the clients.

Step 9: Test your backup.

Make sure that you have configured all of the clients in your backup pool. On all of the clients, execute **cback --full collect**. (You will probably have already tested this command on each of the clients, so it should succeed.)

When all of the client backups have completed, place a valid CD-R or CD-RW disc in your drive, and then use the command **cback --full all**. You should execute this command as root. If the command completes with no output, then the backup was run successfully.

Just to be sure that everything worked properly, check the logfile (`/var/log/cback.log`) on the master and each of the clients, and also mount the CD-R or CD-RW disc on the master to be sure it can be read.

You may also want to run **cback purge** on the master and each client once you have finished validating that everything worked.

If Cedar Backup ever completes “normally” but the disc that is created is not usable, please report this as a bug.⁷ To be safe, always enable the consistency check option in the store configuration section.

Chapter 5. Official Extensions

System Information Extension

The System Information Extension is a simple Cedar Backup extension used to save off important system recovery information that might be useful when reconstructing a “broken” system. It is intended to be run either immediately before or immediately after the standard collect action.

This extension saves off the following information to the configured Cedar Backup collect directory. Saved off data is always compressed using **bzip2**.

- Currently-installed Debian packages via **dpkg --get-selections**
- Disk partition information via **fdisk -l**
- System-wide mounted filesystem contents, via **ls -laR**

The Debian-specific information is only collected on systems where `/usr/bin/dpkg` exists.

To enable this extension, add the following section to the Cedar Backup configuration file:

```
<extensions>
  <action>
    <name>sysinfo</name>
    <module>CedarBackup2.extend.sysinfo</module>
    <function>executeAction</function>
    <index>101</index>
  </action>
</extensions>
```

This extension relies on the options and collect configuration sections in the standard Cedar Backup configuration file, but requires no new configuration of its own.

Subversion Extension

The Subversion Extension is a Cedar Backup extension used to back up Subversion¹ version control repositories via the Cedar Backup command line. It is intended to be run either immediately before or immediately after the standard collect action.

There are two different kinds of Subversion repositories at this writing: BDB (Berkeley Database) and fsfs (a more CVS-like “filesystem within a filesystem”). This extension only works with BDB repositories. Each Subversion repository can be backed using the same collect modes allowed for filesystems in the standard Cedar Backup collect action (weekly, daily, incremental) and the output can be compressed using either **gzip** or **bzip2**.

¹See <http://subversion.org>

To enable this extension, add the following section to the Cedar Backup configuration file:

```
<extensions>
  <action>
    <name>subversion</name>
    <module>CedarBackup2.extend.subversion</module>
    <function>executeAction</function>
    <index>101</index>
  </action>
</extensions>
```

This extension relies on the options and collect configuration sections in the standard Cedar Backup configuration file, and then also requires its own subversion configuration section. This is an example Subversion configuration section:

```
<subversion>
  <collect_mode>incr</collect_mode>
  <compress_mode>none</compress_mode>
  <repository>
    <abs_path>/opt/public/svn/software</abs_path>
    <collect_mode>daily</collect_mode>
  </repository>
  <repository>
    <abs_path>/opt/public/svn/web</abs_path>
    <compress_mode>gzip</compress_mode>
  </repository>
</subversion>
```

The following elements are part of the Subversion configuration section:

collect_mode

Default collect mode.

The collect mode describes how frequently a Subversion repository is backed up. The Subversion extension recognizes the same collect modes as the standard Cedar Backup collect action (see Chapter 2, *Basic Concepts*).

This value is the collect mode that will be used by default during the backup process. Individual repositories (below) may override this value. If *all* individual repositories provide their own value, then this default value may be omitted from configuration.

Note: if your backup device does not support multisession discs, then you should probably use the daily collect mode to avoid losing data.

Restrictions: Must be one of daily, weekly or incr.

compress_mode

Default compress mode.

Subversion repositories backups are just specially-formatted text files, and often compress quite

well using **gzip** or **bzip2**. The compress mode describes how the backed-up data will be compressed, if at all.

This value is the compress mode that will be used by default during the backup process. Individual repositories (below) may override this value. If *all* individual repositories provide their own value, then this default value may be omitted from configuration.

Restrictions: Must be one of none, gzip or bzip2.

repository

A Subversion repository to be collected.

This is a subsection which contains information about a specific Subversion repository to be backed up.

This section can be repeated as many times as is necessary. At least one repository directory must be configured.

The repository subsection contains the following fields:

`abs_path`

Absolute path of the Subversion repository to back up.

Restrictions: Must be an absolute path.

`collect_mode`

Collect mode for this repository.

This field is optional. If it doesn't exist, the backup will use the default collect mode.

Restrictions: Must be one of daily, weekly or incr.

`compress_mode`

Compress mode for this repository.

This field is optional. If it doesn't exist, the backup will use the default compress mode.

Restrictions: Must be one of none, gzip or bzip2.

MySQL Extension

The MySQL Extension is a Cedar Backup extension used to back up MySQL ² databases via the Cedar Backup command line. It is intended to be run either immediately before or immediately after the standard collect action.

The backup is done via the **mysqldump** command included with the MySQL product. Output can be compressed using either **gzip** or **bzip2**. Administrators can configure the extension either to back up all databases or to back up only specific databases. The extension assumes that all databases can be backed up by a single user (typically `root`).

Note that this code always produces a full backup. There is currently no facility for making incremental

²See <http://www.mysql.com>

backups. If/when someone has a need for this and can describe how to do it correctly, I'll update this extension or provide another.

Unfortunately, use of this extension *will* expose usernames and passwords in the process listing (via **ps**) when the backup is running. This is because none of the official MySQL backup scripts provide a good way to specify password other than via the `--password` command-line option. The only workaround I can come up with would be to manipulate program I/O interactively through a pipe, which is a real pain.

To enable this extension, add the following section to the Cedar Backup configuration file:

```
<extensions>
  <action>
    <name>mysql</name>
    <module>CedarBackup2.extend.mysql</module>
    <function>executeAction</function>
    <index>101</index>
  </action>
</extensions>
```

This extension relies on the options and collect configuration sections in the standard Cedar Backup configuration file, and then also requires its own `mysql` configuration section. This is an example MySQL configuration section:

```
<mysql>
  <user>root</user>
  <password>password</password>
  <compress_mode>bzip2</compress_mode>
  <all>Y</all>
</mysql>
```

You should always make `/etc/cback.conf` unreadable to non-root users once you place MySQL configuration into it, since it contains information about available MySQL databases, usernames and passwords.

The following elements are part of the MySQL configuration section:

user

Database user.

The database user that the backup should be executed as. Even if you list more than one database (below) all backups must be done as the same user. Typically, this would be `root` (i.e. the database root user, not the system root user).

Restrictions: Must be non-empty.

password

Password associated with the database user.

Note that once you put this value into configuration, you should make sure that your configuration file cannot be read by users which should not see this password.

Restrictions: Must be non-empty.

`compress_mode`
Compress mode.

MySQL databases dumps are just specially-formatted text files, and often compress quite well using **gzip** or **bzip2**. The compress mode describes how the backed-up data will be compressed, if at all.

Restrictions: Must be one of `none`, `gzip` or `bzip2`.

`all`
Indicates whether to back up all databases.

If this value is `Y`, then all MySQL databases will be backed up. If this value is `N`, then one or more specific databases must be specified (see below).

If you choose this option, the entire database backup will go into one big dump file.

Restrictions: Must be a boolean (`Y` or `N`).

`database`
Named database to be backed up.

If you choose to specify individual databases rather than all databases, then each database will be backed up into its own dump file.

This field can be repeated as many times as is necessary. At least one database must be configured if the `all` option (above) is set to `N`. You may not configure any individual databases if the `all` option is set to `Y`.

Restrictions: Must be non-empty.

Appendix A. Extension Architecture Interface

The Cedar Backup *Extension Architecture Interface* is the application programming interface used by third-party developers to write Cedar Backup extensions. This appendix briefly specifies the interface in enough detail for someone to successfully implement an extension.

You will recall that Cedar Backup extensions are third-party pieces of code which extend Cedar Backup's functionality. Extensions can be invoked from the Cedar Backup command line and are allowed to place their configuration in Cedar Backup's configuration file.

There is a one-to-one mapping between a command-line extended action and an extension function. The mapping is configured in the Cedar Backup configuration file using a section something like this:

```
<extensions>
  <action>
    <name>database</name>
    <module>foobar</module>
    <function>database</function>
    <index>101</index>
  </action>
</extensions>
```

In this case, the action “database” has been mapped to the extension function `foobar.database`.

Extension functions can take any actions they would like to once they have been invoked, but must abide by these rules:

1. Extensions may not write to `stdout` or `stderr` using functions such as `print` or `sys.write`.
2. All logging must take place using the Python logging facility. Flow-of-control logging should happen on the `CedarBackup2.log` topic. Authors can assume that `ERROR` will always go to the terminal, that `INFO` and `WARN` will always be logged, and that `DEBUG` will be ignored unless debugging is enabled.
3. Any time an extension invokes a command-line utility, it must be done through the `CedarBackup2.util.executeCommand` function. This will help keep Cedar Backup safer from format-string attacks, and will make it easier to consistently log command-line process output.
4. Extensions may not return any value.
5. Extensions must throw a Python exception containing a descriptive message if processing fails. Extension authors can use their judgement as to what constitutes failure; however, any problems during execution should result in either a thrown exception or a logged message.
6. Extensions may rely only on Cedar Backup functionality that is advertised as being part of the public interface. This means that extensions cannot directly make use of methods, functions or values starting with the `_` character.

7. Extension authors are encouraged to extend the Cedar Backup public interface through normal methods of inheritance. However, no extension is allowed to directly change Cedar Backup code in a way that would affect how Cedar Backup itself executes when the extension has not been invoked. For instance, extensions would not be allowed to add new command-line options or new writer types.

Extension functions take three arguments: the path to configuration on disk, an `CedarBackup2.cli.Options` object representing the command-line options in effect, and a `CedarBackup2.config.Config` object representing parsed standard configuration.

```
def function(configPath, options, config):
    """Sample extension function."""
    pass
```

This interface is structured so that simple extensions can use standard configuration without having to parse it for themselves, but more complicated extensions can get at the configuration file on disk and parse it again as needed.

The interface to the `CedarBackup2.cli.Options` and `CedarBackup2.config.Config` classes has been thoroughly documented using Epydoc, and the documentation is available on the Cedar Backup website. The interface is guaranteed to change only in backwards-compatible ways unless the Cedar Backup major version number is bumped (i.e. from 2 to 3).

If an extension needs to add its own configuration information to the Cedar Backup configuration file, this extra configuration must be added in a new configuration section using a name that does not conflict with standard configuration or other known extensions.

For instance, our hypothetical database extension might require configuration indicating the path to some repositories to back up. This information might go into a section something like this:

```
<database>
  <repository>/path/to/repo1</repository>
  <repository>/path/to/repo2</repository>
</database>
```

In order to read this new configuration, the extension code can either inherit from the `Config` object and create a subclass that knows how to parse the new `database` config section, or can write its own code to parse whatever it needs out of the file. Either way, the resulting code is completely independent of the standard Cedar Backup functionality.

Appendix B. Dependencies

Python 2.3

Version 2.3 of the Python interpreter was released on 29 July 2003, so most “current” Linux distributions should include it (although Debian “woody” does not include it.)

Source	URL
upstream	http://www.python.org
Debian	http://packages.debian.org/testing/python/python2.3
Gentoo	http://packages.gentoo.org/packages/?category=dev-lang;name=python;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=python

If you can't find a package for your system, install from the package source, using the “upstream” link.

PyXML

Cedar Backup should work with version 0.8.2 or better of this package.

Source	URL
upstream	http://pyxml.sourceforge.net/
Debian	http://packages.debian.org/testing/python/python-xml
Gentoo	http://packages.gentoo.org/packages/?category=dev-python;name=pyxml;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=pyxml

If you can't find a package for your system, install from the package source, using the “upstream” link.

RSH Server and Client

Although Cedar Backup will technically work with any RSH-compatible server and client pair (such as the classic “rsh” client), most users should only use an SSH (secure shell) server and client.

The defacto standard today is OpenSSH. Some systems package the server and the client together, and others package the server and the client separately. Note that *master* nodes need an SSH client, and *client* nodes need to run an SSH server.

Source	URL
upstream	http://www.openssh.com/
Debian	http://packages.debian.org/testing/net/ssh
Gentoo	http://packages.gentoo.org/packages/?category=net-misc;name=openssh;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=openssh

If you can't find SSH client or server packages for your system, install from the package source, using the “upstream” link.

mkisofs

The **mkisofs** command is used create ISO CD images that can later be written to backup media.

Source	URL
upstream	http://freshmeat.net/projects/mkisofs/
Debian	http://packages.debian.org/testing/otherosfs/mkisofs
Gentoo	<i>unknown</i>
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=mkisofs

If you can't find a package for your system, install from the package source, using the “upstream” link.

I have classified Gentoo as “unknown” because I can't find a specific package for that platform. I think that maybe **mkisofs** is part of the cdrtools package (see below), but I'm not sure. Any Gentoo users want to enlighten me?

cdrecord

The **cdrecord** command is used to write ISO images to media in a backup device.

Source	URL
upstream	http://freshmeat.net/projects/cdrecord/
Debian	http://packages.debian.org/testing/otherosfs/cdrecord
Gentoo	http://packages.gentoo.org/packages/?category=app-cdr;name=cdrtools;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=cdrecord

If you can't find a package for your system, install from the package source, using the “upstream” link.

eject and **volname**

The **eject** command is used to open and close the tray on a backup device (if the backup device has a tray). Sometimes, the tray must be opened and closed in order to "reset" the device so it notices recent changes to a disc.

The **volname** command is used to determine the volume name of media in a backup device.

Source	URL
upstream	http://sourceforge.net/projects/eject
Debian	http://packages.debian.org/testing/utils/eject
Gentoo	http://packages.gentoo.org/packages/?category=sys-apps;name=eject;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=eject

If you can't find a package for your system, install from the package source, using the “upstream” link.

mount and **umount**

The **mount** and **umount** commands are used to mount and unmount CD media after it has been written, in order to run a consistency check.

Source	URL
upstream	http://freshmeat.net/projects/util-linux/
Debian	http://packages.debian.org/testing/base/mount
Gentoo	<i>unknown</i>
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=mount

If you can't find a package for your system, install from the package source, using the “upstream” link.

I have classified Gentoo as “unknown” because I can't find a specific package for that platform. It may just be that these two utilities are considered standard, and don't have an independent package of their own. Any Gentoo users want to enlighten me?

Appendix C. Copyright

Copyright (c) 2005
Kenneth J. Pronovici

This work is free; you can redistribute it and/or modify it under the terms of the GNU General Public License (the "GPL"), Version 2, as published by the Free Software Foundation.

For the purposes of the GPL, the "preferred form of modification" for this work is the original Docbook XML text files. If you choose to distribute this work in a compiled form (i.e. if you distribute HTML, PDF or Postscript documents based on the original Docbook XML text files), you must also consider image files to be "source code" if those images are required in order to construct a complete and readable compiled version of the work.

This work 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.

Copies of the GNU General Public License are available from the Free Software Foundation website, <http://www.gnu.org/>. You may also write the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

=====

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

GNU GENERAL PUBLIC LICENSE
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

=====