Cedar Backup Software Manual

Kenneth J. Pronovici

Cedar Backup Software Manual

by Kenneth J. Pronovici

Copyright © 2005-2008 Kenneth J. Pronovici

Table of Contents

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 or UNIX-like 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, *Command Line Tools*
> Discusses the various Cedar Backup command-line tools, including the primary **cback** command.

Chapter 5, *Configuration*
> Provides detailed information about how to configure Cedar Backup.

Chapter 6, *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.

Appendix C, *Data Recovery*
> Cedar Backup provides no facility for restoring backups, assuming the administrator can handle this infrequent task. This appendix provides some notes for administrators to work from.

Appendix D, *Securing Password-less SSH Connections*
> Password-less SSH connections are a necessary evil when remote backup processes need to execute without human interaction. This appendix describes some ways that you can reduce the risk to your backup pool should your master machine be compromised.

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 software package designed to manage system backups for a pool of local and remote
machines. Cedar Backup understands how to back up filesystem data as well as MySQL and
PostgreSQL databases and Subversion repositories. It can also be easily extended to support other kinds
of data sources.

Cedar Backup is focused around weekly backups to a single CD or DVD disc, with the expectation that
the disc will be changed or overwritten at the beginning of each week. If your hardware is new enough
(and almost all hardware is today), Cedar Backup can write multisession discs, allowing you to add
incremental data to a disc on a daily basis.

Besides offering command-line utilities to manage the backup process, Cedar Backup provides a
well-organized library of backup-related functionality, written in the Python programming language.

There are many different backup software implementations out there in the free software 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 or DVD 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, a CVS or Subversion repository, or a
small MySQL database, then Cedar Backup is probably worth your time.

Cedar Backup has been developed on a Debian GNU/Linux system and is primarily supported on
Debian and other Linux systems. However, since it is written in portable Python, it should run without
problems on just about any UNIX-like operating system. In particular, full Cedar Backup functionality is
known to work on Debian and SuSE Linux systems, and client functionality is also known to work on
FreeBSD and Mac OS X systems.

To run a Cedar Backup client, you really just need a working Python installation. To run a Cedar
Backup master, you will also need a set of other executables, most of which are related to building and
writing CD/DVD images. A full list of dependencies is provided in the section called "Installing
Dependencies".

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

---

[1] See "SF Mailing Lists" at http://cedar-backup.sourceforge.net/.

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 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 propogated 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

---

[2] See "SF Bug Tracking" at http://cedar-backup.sourceforge.net/.
[3] See Simon Tatham's excellent bug reporting tutorial: http://www.chiark.greenend.org.uk/~sgtatham/bugs.html .
[4] See http://www.python.org/ .

in learning it, but in retrospect it turned out to be a very good decision. From my perspective, Python has almost all of the strengths of Perl, but few of its inherent weaknesses (I feel that 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 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 package [7] 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 3000 tests, which will help to ensure that quality is maintained as development continues into the future. [8]

---

[5]Debian's stable releases are named after characters in the Toy Story movie.
[6]Epydoc is a Python code documentation tool. See http://epydoc.sourceforge.net/.
[7]See http://docs.python.org/lib/module-logging.html .
[8]Tests 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 a single executable (a Python script). The Python package provides both application-specific code and general utilities that 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/DVD 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 and all rsh-based network connections will take place as the backup user.

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 5, *Configuration* for more information on how Cedar Backup is configured.

> **Warning**
>
> You should be aware that backups to CD/DVD media can probably be read by any user which has permissions to mount the CD/DVD 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. See also the section called "Encrypt Extension".

Data Recovery

Cedar Backup does not include any facility to restore backups. Instead, it assumes that the administrator (using the procedures and references in Appendix C, *Data Recovery*) can handle the task of restoring their own system, using the standard system tools at hand.

If I were to maintain recovery code in Cedar Backup, I would almost certainly end up in one of two situations. Either Cedar Backup would only support simple recovery tasks, and those via an interface a lot like that of the underlying system tools; or Cedar Backup would have to include a hugely complicated interface to support more specialized (and hence useful) recovery tasks like restoring individual files as of a certain point in time. In either case, I would end up trying to maintain critical functionality that would be rarely used, and hence would also be rarely tested by end-users. I am uncomfortable asking anyone to rely on functionality that falls into this category.

My primary goal is to keep the Cedar Backup codebase as simple and focused as possible. I hope you can understand how the choice of providing documentation, but not code, seems to strike the best balance between managing code complexity and providing the functionality that end-users need.

Cedar Backup Pools

---

[1]See http://en.wikipedia.org/wiki/Setuid

There are two kinds of machines in a Cedar Backup pool. One machine (the *master*) has a CD or DVD writer 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.

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*, *initialize* and *rebuild*. All of the various actions are discussed further below.

See Chapter 5, *Configuration* for more information on how a backup run is configured.

Flexibility

Cedar Backup was designed to be flexible. It allows you to decide for yourself which backup steps you care about executing (and when you execute them), based on your own situation and your own priorities.

As an example, I always back up every machine I own. I typically keep 7-10 days of staging directories around, but switch CD/DVD media mostly every week. That way, I can periodically take a disc off-site in case the machine gets stolen or damaged.

If you're not worried about these risks, then there's no need to write to disc. In fact, some users prefer to use their master machine as a simple "consolidation point". They don't back up any data on the master, and don't write to disc at all. They just use Cedar Backup to handle the mechanics of moving backed-up data to a central location. This isn't quite what Cedar Backup was written to do, but it is flexible enough to meet their needs.

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 patterns [3] to be excluded. You can even configure a backup "link farm" rather than explicitly listing files and directories in configuration.

This action is optional on the master. You only need to configure and execute the collect action on the master if you have data to back up on that machine. If you plan to use the master only as a "consolidation point" to collect data from other machines, then there is no need to execute the collect action there. If you run the collect action on the master, it behaves the same there as anywhere else, and you have to stage the master's collected data just like any other client (typically by configuring a local peer in the stage action).

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.

For the purposes of this action, the master node can be configured to treat itself as a client node. If you intend to back up data on the master, configure the master as a local peer. Otherwise, just configure each of the clients as a remote peer.

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 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.

Keep in mind that Cedar Backup is flexible about what actions must be executed as part of a backup. If you would prefer, you can stop the backup process at this step, and skip the store step. In this case, the staged directories will represent your backup rather than a disc.

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.

---

[2] Analagous to `.cvsignore` in CVS
[3] In terms of Python regular expressions

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.

This action is entirely optional. If you would prefer to just stage backup data from a set of peers to a master machine, and have the staged directories represent your backup rather than a disc, this is fine.

> ## Warning
>
> The store action is not supported on the Mac OS X (darwin) platform. On that platform, the "automount" function of the Finder interferes significantly with Cedar Backup's ability to mount and unmount media and write to the CD or DVD hardware. The Cedar Backup writer and image functionality works on this platform, but the effort required to fight the operating system about who owns the media and the device makes it nearly impossible to execute the store action successfully.

> **Current Staging Directory**
>
> The store action tries to be smart about finding the current staging directory. It first checks the current day's staging directory. If that directory exists, and it has not yet been written to disc (i.e. there is no store indicator), then it will be used. Otherwise, the store action will look for an unused staging directory for either the previous day or the next day, in that order. A warning will be written to the log under these circumstances (controlled by the <warn_midnite> configuration value).
>
> This behavior varies slightly when the `--full` option is in effect. Under these circumstances, any existing store indicator will be ignored. Also, the store action will always attempt to use the current day's staging directory, ignoring any staging directories for the previous day or the next day. This way, running a full store action more than once concurrently will always produce the same results. (You might imagine a use case where a person wants to make several copies of the same full backup.)

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 *cannot* 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. [4]

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 Initialize Action

The initialize action is used to initialize media for use with Cedar Backup. This is an optional step. By default, Cedar Backup does not need to use initialized media and will write to whatever media exists in the writer device.

However, if the "check media" store configuration option is set to true, Cedar Backup will check the media before writing to it and will error out if the media has not been initialized.

Initializing the media consists of writing a mostly-empty image using a known media label (the media label will begin with "CEDAR BACKUP").

Note that only rewritable media (CD-RW, DVD+RW) can be initialized. It doesn't make any sense to initialize media that cannot be rewritten (CD-R, DVD+R), since Cedar Backup would then not be able to use that media for a backup. You can still configure Cedar Backup to check non-rewritable media; in this case, the check will also pass if the media is apparently unused (i.e. has no media label).

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 other 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.

---

[4]Some users find this surprising, because extensions are configured with sequence numbers. I did it this way because I felt that running extensions as part of the all action would sometimes result in surprising behavior. I am not planning to change the way this works.

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.

Managed Backups

Cedar Backup also supports an optional feature called the "managed backup". This feature is intended for use with remote clients where cron is not available (for instance, SourceForge shell accounts).

When managed backups are enabled, managed clients must still be configured as usual. However, rather than using a cron job on the client to execute the collect and purge actions, the master executes these actions on the client via a remote shell.

To make this happen, first set up one or more managed clients in Cedar Backup configuration. Then, invoke Cedar Backup with the **--managed** command-line option. Whenever Cedar Backup invokes an action locally, it will invoke the same action on each of the managed clients.

Technically, this feature works for any client, not just clients that don't have cron available. Used this way, it can simplify the setup process, because cron only has to be configured on the master. For some users, that may be motivation enough to use this feature all of the time.

However, please keep in mind that this feature depends on a stable network. If your network connection drops, your backup will be interrupted and will not be complete. It is even possible that some of the Cedar Backup metadata (like incremental backup state) will be corrupted. The risk is not high, but it is something you need to be aware of if you choose to use this optional feature.

Media and Device Types

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

When using a new enough backup device, a new "multisession" ISO image [6] 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 Cedar Backup runs. This way, the media is complete and usable at the end of every backup run, but a

---

[5]My original backup device was an old Sony CRX140E 4X CD-RW drive. It has since died, and I currently develop using a Lite-On 1673S DVD±RW drive.
[6]An *ISO image* is the standard way of creating a filesystem to be copied to a CD or DVD. It is essentially a "filesystem-within-a-file" and many UNIX operating 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.

single disc can be used all week long. If your backup device does not support multisession images — which is really unusual today — 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 CD media:

cdr-74
    74-minute non-rewritable CD media

cdrw-74
    74-minute rewritable CD media

cdr-80
    80-minute non-rewritable CD media

cdrw-80
    80-minute rewritable CD media

I have chosen to support just these four types of CD media because they seem to be the most "standard" of the various types commonly sold in the U.S. as of this writing (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.

Cedar Backup also supports two kinds of DVD media:

dvd+r
    Single-layer non-rewritable DVD+R media

dvd+rw
    Single-layer rewritable DVD+RW media

The underlying **growisofs** utility does support other kinds of media (including DVD-R, DVD-RW and BlueRay) which work somewhat differently than standard DVD+R and DVD+RW media. I don't support these other kinds of media because I haven't had any opportunity to work with them. The same goes for dual-layer media of any type.

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 value [7] for each backed-up file. The next time an incremental backup is run, Cedar Backup checks its list of

---

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

file/checksum pairs for each file that might be backed up. If the file's checksum value does not match the 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 which can be executed from the **cback** command line like any other action.

Hopefully, as the Cedar Backup 2.0 user community 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 5, *Configuration* for more information on how extensions are configured, and Chapter 6, *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 or you are running some other platform like FreeBSD or Mac OS X, 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

Cedar Backup has been developed on a Debian GNU/Linux system and is primarily supported on Debian and other Linux systems. However, since it is written in portable Python, it should run without problems on just about any UNIX-like operating system. In particular, full Cedar Backup functionality is known to work on Debian and SuSE Linux systems, and client functionality is also known to work on FreeBSD and Mac OS X systems.

To run a Cedar Backup client, you really just need a working Python installation. To run a Cedar Backup master, you will also need a set of other executables, most of which are related to building and writing CD/DVD images. A full list of dependencies is provided further on in this chapter.

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

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

If you are running a Debian release which contains Cedar Backup, you can use your normal Debian mirror as an APT data source. (The Debian "etch" release is the first release to contain Cedar Backup.) Otherwise, you need to install from the Cedar Solutions APT data source. To do this, add the Cedar Solutions APT data source to your `/etc/apt/sources.list` file. [2]

After you have configured the proper APT data source, install Cedar Backup using this set of commands:

```
$ apt-get update
```

---

[1]See "SF Mailing Lists" at http://cedar-backup.sourceforge.net/.
[2]See "SF Bug Tracking" at http://cedar-backup.sourceforge.net/.

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

Several of the Cedar Backup dependencies are listed as "recommended" rather than required. If you are installing Cedar Backup on a master machine, you must install some or all of the recommended dependencies, depending on which actions you intend to execute. The stage action normally requires ssh, and the store action requires eject and either cdrecord/mkisofs or dvd+rw-tools. Clients must also install some sort of ssh server if a remote master will collect backups from them.

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 these files files in the Cedar Solutions APT source. [3]

In either case, once the package has been installed, you can proceed to configuration as described in Chapter 5, *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] You will have to manage dependencies on your own.

Tip

Many UNIX-like distributions provide an automatic or semi-automatic way to install packages like the ones Cedar Backup requires (think RPMs for Mandrake or RedHat, Gentoo's Portage system, the Fink project for Mac OS X, or the BSD ports 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 and BSD distributions should include it. You must install Python on every peer node in a pool (master or client).

---

[3]See http://cedar-solutions.com/debian.html.
[4]See http://docs.python.org/lib/module-distutils.html .

Additionally, remote client peer nodes must be running an *RSH-compatible* server, such as the **ssh** server, and master nodes must have an RSH-compatible client installed if they need to connect to remote peer machines.

Master machines also require several other system utilities, most having to do with writing and validating CD/DVD media. On master machines, you must make sure that these utilities are available if you want to to run the store action:

- **mkisofs**

- **eject**

- **mount**

- **unmount**

- **volname**

Then, you need this utility if you are writing CD media:

- **cdrecord**

or these utilities if you are writing DVD media:

- **growisofs**

All of these utilities are common and are easy to find for almost any UNIX-like operating system.

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`.

You may also wish to run the unit tests before actually installing anything. Run them like so:

```
python util/test.py
```

If any unit test reports a failure on your system, please email me the output from the unit test, so I can fix the problem. [5] This is particularly important for non-Linux platforms where I do not have a test system available to me.

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 5, *Configuration*.

---

[5] `<support@cedar-solutions.com>`

Chapter 4. Command Line Tools

Overview

Cedar Backup comes with two command-line programs, the **cback** and **cback-span** commands. The **cback** command is the primary command line interface and the only Cedar Backup program that most users will ever need.

Users that have a *lot* of data to back up — more than will fit on a single CD or DVD — can use the interactive **cback-span** tool to split their data between multiple discs.

The cback command

Introduction

Cedar Backup's primary command-line interface is the **cback** command. It controls the entire backup process.

Syntax

The **cback** command 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
  -M, --managed       Include managed clients when executing actions
  -N, --managed-only  Include ONLY managed clients when executing actions
  -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. cdrecord) 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
  -D, --diagnostics   Print runtime diagnostics to the screen and exit

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
```

```
    initialize          Initialize media for use with Cedar Backup
```

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",
"validate", and "initialize" actions may not be combined with
other actions.

Note that the all action *only* executes the standard four actions. It never executes any of the configured extensions. [1]

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.

-M, --managed
   Include managed clients when executing actions. If the action being executed is listed as a managed action for a managed client, execute the action on that client after executing the action locally.

-N, --managed-only
   Include *only* managed clients when executing actions. If the action being executed is listed as a managed action for a managed client, execute the action on that client — but *do not* execute the action locally.

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

---

[1]Some users find this surprising, because extensions are configured with sequence numbers. I did it this way because I felt that running extensions as part of the all action would sometimes result in "surprising" behavior. Better to be definitive than confusing.

-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** command 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** command 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/DVD 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.

-D, --diagnostics

Display runtime diagnostic information and then exit. This diagnostic information is often useful when filing 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`, `validate`, or `initialize` 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.

The cback-span command

Introduction

Cedar Backup was designed — and is still primarily focused — around weekly backups to a single CD or DVD. Most users who back up more data than fits on a single disc seem to stop their backup process

at the stage step, using Cedar Backup as an easy way to collect data.

However, some users have expressed a need to write these large kinds of backups to disc — if not every day, then at least occassionally. The **cback-span** tool was written to meet those needs. If you have staged more data than fits on a single CD or DVD, you can use **cback-span** to split that data between multiple discs.

cback-span is not a general-purpose disc-splitting tool. It is a specialized program that requires Cedar Backup configuration to run. All it can do is read Cedar Backup configuration, find any staging directories that have not yet been written to disc, and split the files in those directories between discs.

cback-span accepts many of the same command-line options as **cback**, but *must* be run interactively. It cannot be run from cron. This is intentional. It is intended to be a useful tool, not a new part of the backup process (that is the purpose of an extension).

In order to use **cback-span**, you must configure your backup such that the largest individual backup file can fit on a single disc. *The command will not split a single file onto more than one disc.* All it can do is split large directories onto multiple discs. Files in those directories will be arbitrarily split up so that space is utilized most efficiently.

Syntax

The **cback-span** command has the following syntax:

```
Usage: cback-span [switches]

Cedar Backup 'span' tool.

This Cedar Backup utility spans staged data between multiple discs.
It is a utility, not an extension, and requires user interaction.

The following switches are accepted, mostly to set up underlying
Cedar Backup functionality:

  -h, --help     Display this usage/help listing
  -V, --version  Display version information
  -b, --verbose  Print verbose output as well as logging to disk
  -c, --config   Path to config file (default: /etc/cback.conf)
  -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. cdrecord) 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
```

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.

-c, --config

Specify the path to an alternate configuration file. The default configuration file is /etc/cback.conf.

-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** command 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** command 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/DVD 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.

Using cback-span

As discussed above, the **cback-span** is an interactive command. It cannot be run from cron.

You can typically use the default answer for most questions. The only two questions that you may not want the default answer for are the fit algorithm and the cushion percentage.

The cushion percentage is used by **cback-span** to determine what capacity to shoot for when splitting up your staging directories. A 650 MB disc does not fit fully 650 MB of data. It's usually more like 627 MB of data. The cushion percentage tells **cback-span** how much overhead to reserve for the filesystem. The default of 4% is usually OK, but if you have problems you may need to increase it slightly.

The fit algorithm tells **cback-span** how it should determine which items should be placed on each disc.

If you don't like the result from one algorithm, you can reject that solution and choose a different algorithm.

The four available fit algorithms are:

worst
: The *worst-fit* algorithm.

The worst-fit algorithm proceeds through a sorted list of items (sorted from smallest to largest) until running out of items or meeting capacity exactly. If capacity is exceeded, the item that caused capacity to be exceeded is thrown away and the next one is tried. The algorithm effectively includes the maximum number of items possible in its search for optimal capacity utilization. It tends to be somewhat slower than either the best-fit or alternate-fit algorithm, probably because on average it has to look at more items before completing.

best
: The *best-fit* algorithm.

The best-fit algorithm proceeds through a sorted list of items (sorted from largest to smallest) until running out of items or meeting capacity exactly. If capacity is exceeded, the item that caused capacity to be exceeded is thrown away and the next one is tried. The algorithm effectively includes the minimum number of items possible in its search for optimal capacity utilization. For large lists of mixed-size items, it's not unusual to see the algorithm achieve 100% capacity utilization by including fewer than 1% of the items. Probably because it often has to look at fewer of the items before completing, it tends to be a little faster than the worst-fit or alternate-fit algorithms.

first
: The *first-fit* algorithm.

The first-fit algorithm proceeds through an unsorted list of items until running out of items or meeting capacity exactly. If capacity is exceeded, the item that caused capacity to be exceeded is thrown away and the next one is tried. This algorithm generally performs more poorly than the other algorithms both in terms of capacity utilization and item utilization, but can be as much as an order of magnitude faster on large lists of items because it doesn't require any sorting.

alternate
: A hybrid algorithm that I call *alternate-fit*.

This algorithm tries to balance small and large items to achieve better end-of-disk performance. Instead of just working one direction through a list, it alternately works from the start and end of a sorted list (sorted from smallest to largest), throwing away any item which causes capacity to be exceeded. The algorithm tends to be slower than the best-fit and first-fit algorithms, and slightly faster than the worst-fit algorithm, probably because of the number of items it considers on average before completing. It often achieves slightly better capacity utilization than the worst-fit algorithm, while including slightly fewer items.

Sample run

Below is a log showing a sample **cback-span** run.

```
================================================
          Cedar Backup 'span' tool
```

21

```
=================================================

This the Cedar Backup span tool.  It is used to split up staging
data when that staging data does not fit onto a single disc.

This utility operates using Cedar Backup configuration.  Configuration
specifies which staging directory to look at and which writer device
and media type to use.

Continue? [Y/n]:
===

Cedar Backup store configuration looks like this:

    Source Directory...: /tmp/staging
    Media Type.........: cdrw-74
    Device Type........: cdwriter
    Device Path........: /dev/cdrom
    Device SCSI ID.....: None
    Drive Speed........: None
    Check Data Flag....: True
    No Eject Flag......: False

Is this OK? [Y/n]:
===

Please wait, indexing the source directory (this may take a while)...
===

The following daily staging directories have not yet been written to disc:

    /tmp/staging/2007/02/07
    /tmp/staging/2007/02/08
    /tmp/staging/2007/02/09
    /tmp/staging/2007/02/10
    /tmp/staging/2007/02/11
    /tmp/staging/2007/02/12
    /tmp/staging/2007/02/13
    /tmp/staging/2007/02/14

The total size of the data in these directories is 1.00 GB.

Continue? [Y/n]:
===

Based on configuration, the capacity of your media is 650.00 MB.

Since estimates are not perfect and there is some uncertainly in
media capacity calculations, it is good to have a "cushion",
a percentage of capacity to set aside.  The cushion reduces the
capacity of your media, so a 1.5% cushion leaves 98.5% remaining.

What cushion percentage? [4.00]:
===

The real capacity, taking into account the 4.00% cushion, is 627.25 MB.
It will take at least 2 disc(s) to store your 1.00 GB of data.

Continue? [Y/n]:
===
```

Which algorithm do you want to use to span your data across
multiple discs?

The following algorithms are available:

    first....: The "first-fit" algorithm
    best.....: The "best-fit" algorithm
    worst....: The "worst-fit" algorithm
    alternate: The "alternate-fit" algorithm

If you don't like the results you will have a chance to try a
different one later.

Which algorithm? [worst]:
===

Please wait, generating file lists (this may take a while)...
===

Using the "worst-fit" algorithm, Cedar Backup can split your data
into 2 discs.

Disc 1: 246 files, 615.97 MB, 98.20% utilization
Disc 2: 8 files, 412.96 MB, 65.84% utilization

Accept this solution? [Y/n]: n
===

Which algorithm do you want to use to span your data across
multiple discs?

The following algorithms are available:

    first....: The "first-fit" algorithm
    best.....: The "best-fit" algorithm
    worst....: The "worst-fit" algorithm
    alternate: The "alternate-fit" algorithm

If you don't like the results you will have a chance to try a
different one later.

Which algorithm? [worst]: alternate
===

Please wait, generating file lists (this may take a while)...
===

Using the "alternate-fit" algorithm, Cedar Backup can split your data
into 2 discs.

Disc 1: 73 files, 627.25 MB, 100.00% utilization
Disc 2: 181 files, 401.68 MB, 64.04% utilization

Accept this solution? [Y/n]: y
===

Please place the first disc in your backup device.
Press return when ready.
===

```
Initializing image...
Writing image to disc...
```

Chapter 5. 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 "The **cback** command" (in Chapter 4, *Command Line Tools*) 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 Platform?

Cedar Backup has been designed for use on all UNIX-like systems. However, since it was developed on a Debian GNU/Linux 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 platform you are running (i.e. RedHat, Gentoo, FreeBSD, etc.). If instructions vary for a particular platform, you will find a note related to that platform.

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

Configuration File Format

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

---

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

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.

Note

Even though the Mac OS X (darwin) filesystem is *not* case-sensitive, Cedar Backup configuration *is* generally case-sensitive on that platform, just like on all other platforms. For instance, even though the files "Ken" and "ken" might be the same on the Mac OS X filesystem, an exclusion in Cedar Backup configuration for "ken" will only match the file if it is actually on the filesystem with a lower-case "k" as its first letter. This won't surprise the typical UNIX user, but might surprise someone who's gotten into the "Mac Mindset".

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/examples/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>
   <peers>
      <peer>
         <name>debian</name>
         <type>local</type>
         <collect_dir>/opt/backup/collect</collect_dir>
      </peer>
   </peers>
   <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>
```

---

[2]See the section called "The Backup Process", in Chapter 2, *Basic Concepts*.

```
        <file>
            <abs_path>/home/root/.profile</abs_path>
            <collect_mode>weekly</collect_mode>
        </file>
    </collect>
    <stage>
        <staging_dir>/opt/backup/staging</staging_dir>
    </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>
        <target_scsi_id>0,0,0</target_scsi_id>
        <drive_speed>4</drive_speed>
        <check_data>Y</check_data>
        <check_media>Y</check_media>
        <warn_midnite>Y</warn_midnite>
    </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.

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>
   <rsh_command>/usr/bin/ssh</rsh_command>
   <cback_command>/usr/bin/cback</cback_command>
   <managed_actions>collect, purge</managed_actions>
   <override>
      <command>cdrecord</command>
      <abs_path>/opt/local/bin/cdrecord</abs_path>
   </override>
   <override>
      <command>mkisofs</command>
      <abs_path>/opt/local/bin/mkisofs</abs_path>
   </override>
   <pre_action_hook>
      <action>collect</action>
      <command>echo "I AM A PRE-ACTION HOOK RELATED TO COLLECT"</command>
   </pre_action_hook>
   <post_action_hook>
      <action>collect</action>
      <command>echo "I AM A POST-ACTION HOOK RELATED TO COLLECT"</command>
   </post_action_hook>
</options>
```

The following elements are part of the options configuration section:

starting_day
  Day that starts the week.

28

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 filesystem 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 filesystem 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.

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. Technically, this value is not needed by clients, but we require it for all config files anyway.

Restrictions: Must be non-empty

`rsh_command`
Default rsh-compatible command to use for remote shells.

The rsh command should be the exact command used for remote shells, including any required options.

This value is used as the default value for all managed clients. It is optional, because it is only used when executing actions on managed clients. However, each managed client must either be able to

read the value from options configuration or must set the value explicitly.

Restrictions: Must be non-empty

cback_command
Default cback-compatible command to use on managed remote clients.

The cback command should be the exact command used for for executing **cback** on a remote managed client, including any required command-line options. Do *not* list any actions in the command line, and do *not* include the **--full** command-line option.

This value is used as the default value for all managed clients. It is optional, because it is only used when executing actions on managed clients. However, each managed client must either be able to read the value from options configuration or must set the value explicitly.

Note: if this command-line is complicated, it is often better to create a simple shell script on the remote host to encapsulate all of the options. Then, just reference the shell script in configuration.

Restrictions: Must be non-empty

managed_actions
Default set of actions that are managed on remote clients.

This is a comma-separated list of actions that the master will manage on behalf of remote clients. Typically, it would include only collect-like actions and purge.

This value is used as the default value for all managed clients. It is optional, because it is only used when executing actions on managed clients. However, each managed client must either be able to read the value from options configuration or must set the value explicitly.

Restrictions: Must be non-empty.

override
Command to override with a customized path.

This is a subsection which contains a command to override with a customized path. This functionality would be used if root's $PATH does not include a particular required command, or if there is a need to use a version of a command that is different than the one listed on the $PATH. Most users will only use this section when directed to, in order to fix a problem.

This section is optional, and can be repeated as many times as necessary.

This subsection must contain the following two fields:

command
Name of the command to be overridden, i.e. "cdrecord".

Restrictions: Must be a non-empty string.

abs_path
The absolute path where the overridden command can be found.

Restrictions: Must be an absolute path.

pre_action_hook

Hook configuring a command to be executed before an action.

This is a subsection which configures a command to be executed immediately before a named action. It provides a way for administrators to associate their own custom functionality with standard Cedar Backup actions or with arbitrary extensions.

This section is optional, and can be repeated as many times as necessary.

This subsection must contain the following two fields:

`action`
Name of the Cedar Backup action that the hook is associated with. The action can be a standard backup action (collect, stage, etc.) or can be an extension action. No validation is done to ensure that the configured action actually exists.

Restrictions: Must be a non-empty string.

`command`
Name of the command to be executed. This item can either specify the path to a shell script of some sort (the recommended approach) or can include a complete shell command.

Note: if you choose to provide a complete shell command rather than the path to a script, you need to be aware of some limitations of Cedar Backup's command-line parser. You cannot use a subshell (via the `` `command` `` or `$(command)` syntaxes) or any shell variable in your command line. Additionally, the command-line parser only recognizes the double-quote character (`"`) to delimit groupings or strings on the command-line. The bottom line is, you are probably best off writing a shell script of some sort for anything more sophisticated than very simple shell commands.

Restrictions: Must be a non-empty string.

`post_action_hook`
Hook configuring a command to be executed after an action.

This is a subsection which configures a command to be executed immediately after a named action. It provides a way for administrators to associate their own custom functionality with standard Cedar Backup actions or with arbitrary extensions.

This section is optional, and can be repeated as many times as necessary.

This subsection must contain the following two fields:

`action`
Name of the Cedar Backup action that the hook is associated with. The action can be a standard backup action (collect, stage, etc.) or can be an extension action. No validation is done to ensure that the configured action actually exists.

Restrictions: Must be a non-empty string.

`command`
Name of the command to be executed. This item can either specify the path to a shell script of some sort (the recommended approach) or can include a complete shell command.

Note: if you choose to provide a complete shell command rather than the path to a script, you need to be aware of some limitations of Cedar Backup's command-line parser. You cannot use a subshell (via the `` `command` `` or `$(command)` syntaxes) or any shell variable in your command line. Additionally, the command-line parser only recognizes the double-quote character (`"`) to delimit

groupings or strings on the command-line. The bottom line is, you are probably best off writing a shell script of some sort for anything more sophisticated than very simple shell commands.

Restrictions: Must be a non-empty string.

Peers Configuration

The peers configuration section contains a list of the peers managed by a master. This section is only required on a master.

This is an example peers configuration section:

```
<peers>
   <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>
   <peer>
      <name>machine3</name>
      <type>remote</type>
      <managed>Y</managed>
      <backup_user>backup</backup_user>
      <collect_dir>/opt/backup/collect</collect_dir>
      <rcp_command>/usr/bin/scp</rcp_command>
      <rsh_command>/usr/bin/ssh</rsh_command>
      <cback_command>/usr/bin/cback</cback_command>
      <managed_actions>collect, purge</managed_actions>
   </peer>
</peers>
```

The following elements are part of the peers configuration section:

`peer` (local version)
   Local client peer in a backup pool.

   This is a subsection which contains information about a specific local client peer managed by a master.

   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, and unique among all peers.

`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 managed by a master. A remote peer is one which can 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, and unique among all peers.

`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`.

`managed`
Indicates whether this peer is managed.

A managed peer (or managed client) is a peer for which the master manages all of the backup activites via a remote shell.

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

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

`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.

`rsh_command`
The rsh-compatible command for this peer.

The rsh command should be the exact command used for remote shells, including any required options.

This value only applies if the peer is managed.

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

Restrictions: Must be non-empty

`cback_command`
The cback-compatible command for this peer.

The cback command should be the exact command used for for executing cback on the peer as part of a managed backup. This value must include any required command-line options. Do *not* list any actions in the command line, and do *not* include the **--full** command-line option.

This value only applies if the peer is managed.

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

options section.

Note: if this command-line is complicated, it is often better to create a simple shell script on the remote host to encapsulate all of the options. Then, just reference the shell script in configuration.

Restrictions: Must be non-empty

```
managed_actions
```
Set of actions that are managed for this peer.

This is a comma-separated list of actions that the master will manage on behalf this peer. Typically, it would include only collect-like actions and purge.

This value only applies if the peer is managed.

This field is optional. if it doesn't exist, the backup will use the default list of managed actions from the options section.

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 and/or files to collect. You can also configure an ignore indicator file, which lets users mark their own directories as not backed up.

Using a Link Farm

Sometimes, it's not very convenient to list directories one by one in the Cedar Backup configuration file. For instance, when backing up your home directory, you often exclude as many directories as you include. The ignore file mechanism can be of some help, but it still isn't very convenient if there are a lot of directories to ignore (or if new directories pop up all of the time).

In this situation, one option is to use a *link farm* rather than listing all of the directories in configuration. A link farm is a directory that contains nothing but a set of soft links to other files and directories. Normally, Cedar Backup does not follow soft links, but you can override this behavior for individual directories using the `link_depth` and `dereference` options (see below).

When using a link farm, you still have to deal with each backed-up directory individually, but you don't have to modify configuration. Some users find that this works better for them.

In order to actually execute the collect action, you must have configured at least one collect directory or one collect file. However, if you are only including collect configuration for use by an extension, then it's OK to leave out these sections. The validation will take place only when the collect action is executed.

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>
    <file>
        <abs_path>/home/root/.profile</abs_path>
    </file>
    <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 suppport 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

`file`
A file to be collected.

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

This section can be repeated as many times as is necessary. At least one collect directory or collect file must be configured when the collect action is executed.

The collect file subsection contains the following fields:

`abs_path`
Absolute path of the file to collect.

Restrictions: Must be an absolute path.

`collect_mode`
Collect mode for this file

The collect mode describes how frequently a file 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 suppport 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 file.

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`.

---

[3]See http://docs.python.org/lib/re-syntax.html

`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 or collect file must be configured when the collect action is executed.

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.

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 suppport 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.

39

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

link_depth
Link depth value to use for this directory.

The link depth is maximum depth of the tree at which soft links should be followed. So, a depth of 0 does not follow any soft links within the collect directory, a depth of 1 follows only links immediately within the collect directory, a depth of 2 follows the links at the next level down, etc.

This field is optional. If it doesn't exist, the backup will assume a value of zero, meaning that soft links within the collect directory will never be followed.

Restrictions: If set, must be an integer # 0.

dereference
Whether to dereference soft links.

If this flag is set, links that are being followed will be dereferenced before being added to the backup. The link will be added (as a link), and then the directory or file that the link points at will be added as well.

This value only applies to a directory where soft links are being followed (per the link_depth configuration option). It never applies to a configured collect directory itself, only to other directories within the collect directory.

This field is optional. If it doesn't exist, the backup will assume that links should never be dereferenced.

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

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.

40

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 indicates where date from peers can be staged to.

This section can also (optionally) override the list of peers so that not all peers are staged. If you provide *any* peers in this section, then the list of peers here completely replaces the list of peers in the peers configuration section for the purposes of staging.

This is an example stage configuration section for the simple case where the list of peers is taken from peers configuration:

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

This is an example stage configuration section that overrides the default list of peers:

```
<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 requires 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.

    Remember, if you provide *any* local or remote peer in staging configuration, the global peer configuration is completely replaced by the staging peer configuration.

    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, and unique among all peers.

`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.

Remember, if you provide *any* local or remote peer in staging configuration, the global peer configuration is completely replaced by the staging peer configuration.

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, and unique among all peers.

`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>
    <check_media>Y</check_media>
    <warn_midnite>Y</warn_midnite>
    <no_eject>N</no_eject>
    <blank_behavior>
        <mode>weekly</mode>
        <factor>1.3</factor>
    </blank_behavior>
</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

device_type
Type of the device used to write the media.

This field controls which type of writer device will be used by Cedar Backup. Currently, Cedar Backup supports CD writers (`cdwriter`) and DVD writers (`dvdwriter`).

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

Restrictions: If set, must be either `cdwriter` or `dvdwriter`.

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.

You must choose a media type that is appropriate for the device type you chose above. 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` if device type is `cdwriter`; or one of `dvd+r` or `dvd+rw` if device type is `dvdwriter`.

target_device
Filesystem device name for writer device.

This value is required for both CD writers and DVD writers.

This is the UNIX device name for the writer drive, for instance `/dev/scd0` or `/dev/cdrw`.

In some cases, this device name is used to directly write to media. This is true all of the time for DVD writers, and is true for CD writers when a SCSI id (see below) has not been specified.

Besides this, the device name is also 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 the writer device.

This value is optional for CD writers and is ignored for DVD writers.

If you have configured your CD writer hardware to work through the normal filesystem device path, then you can leave this parameter unset. Cedar Backup will just use the target device (above) when talking to **cdrecord**.

Otherwise, if you have SCSI CD writer hardware or you have configured your non-SCSI hardware

45

to operate like a SCSI device, then you need to provide Cedar Backup with a SCSI id it can use when talking with **cdrecord**.

For the purposes of Cedar Backup, a valid SCSI identifier must either be in the standard SCSI identifier form `scsibus,target,lun` or in the specialized-method form `<method>:scsibus,target,lun`.

An example of a standard SCSI identifier is `1,6,2`. Today, the two most common examples of the specialized-method form are `ATA:scsibus,target,lun` and `ATAPI:scsibus,target,lun`, but you may occassionally see other values (like `OLDATAPI` in some forks of **cdrecord**).

See the section called "Configuring your Writer Device" for more information on writer devices and how they are configured.

Restrictions: If set, 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.

For DVD writers, it is best to leave this value unset, so **growisofs** can pick an appropriate speed. For CD writers, since media can be speed-sensitive, it is probably best to set a sensible value based on your specific writer and media.

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

check_data

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 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`).

check_media

Whether the media should be checked before writing to it.

By default, Cedar Backup does not check its media before writing to it. It will write to any media in the backup device. If you set this flag to Y, Cedar Backup will make sure that the media has been initialized before writing to it. (Rewritable media is initialized using the initialize action.)

If the configured media is not rewritable (like CD-R), then this behavior is modified slightly. For this kind of media, the check passes either if the media has been initialized *or* if the media appears unused.

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

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

warn_midnite
:   Whether to generate warnings for crossing midnite.

     This field indicates whether warnings should be generated if the store operation has to cross a midnite boundary in order to find data to write to disc. For instance, a warning would be generated if valid store data was only found in the day before or day after the current day.

     Configuration for some users is such that the store operation will always cross a midnite boundary, so they will not care about this warning. Other users will expect to never cross a boundary, and want to be notified that something "strange" might have happened.

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

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

no_eject
:   Indicates that the writer device should not be ejected.

     Under some circumstances, Cedar Backup ejects (opens and closes) the writer device. This is done because some writer devices need to re-load the media before noticing a media state change (like a new session).

     For most writer devices this is safe, because they have a tray that can be opened and closed. If your writer device does not have a tray *and* Cedar Backup does not properly detect this, then set this flag. Cedar Backup will not ever issue an eject command to your writer.

     Note: this could cause problems with your backup. For instance, with many writers, the check data step may fail if the media is not reloaded first. If this happens to you, you may need to get a different writer device.

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

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

blank_behavior
:   Optimized blanking strategy.

     For more information about Cedar Backup's optimized blanking strategy, see the section called "Optimized Blanking Stategy".

     This entire configuration section is optional. However, if you choose to provide it, you must configure both a blanking mode and a blanking factor.

     blank_mode
     Blanking mode.

     *Restrictions:*Must be one of "daily" or "weekly".

     blank_factor
     Blanking factor.

*Restrictions:*Must be a floating point number # 0.

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.

48

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.

Warning

Extended actions should always be configured to run *before* the standard action they are associated with. This is because of the way indicator files are used in Cedar Backup. For instance, the staging process considers the collect action to be complete for a peer if the file cback.collect can be found in that peer's collect directory.

If you were to run the standard collect action before your other collect-like actions, the indicator file would be written after the collect action completes but *before* all of the other actions even run. Because of this, there's a chance the stage process might back up the collect directory before the entire set of collect-like actions have completed — and you would get no warning about this in your email!

So, 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 "foo.bar()". You think of this backup as a "collect" kind of action, so you want it to be performed immediately before the collect action.

To configure this extension, you would list an action with a name "database", a module "foo", a function name "bar" and an index of "99".

This is how the hypothetical action would be configured:

```
<extensions>
   <action>
      <name>database</name>
      <module>foo</module>
      <function>bar</function>
      <index>99</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.

Tip

This setup procedure discusses how to set up Cedar Backup in the "normal case" for a pool of one. If you would like to modify the way Cedar Backup works (for instance, by ignoring the store stage and just letting your backup sit in a staging directory), you can do that. You'll just have to modify the procedure below based on information in the remainder of the manual.

Step 1: Decide when you will run your backup.

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 set of cron jobs. Although you won't create your cron jobs just yet, you should decide now when you will run your backup so you are prepared for later.

Backing up large directories and creating ISO filesystem 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.

Warning

Because of the way Cedar Backup works, you must ensure that your backup *always* runs son the first day of your configured week. This is because Cedar Backup will only clear incremental backup information and re-initialize your media when running on the first day of the week. If you skip running Cedar Backup on the first day of the week, your backups will likely be "confused" until the next week begins, or until you re-run the backup using the `--full` flag.

Step 2: 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 3: Configure your writer device.

Before using Cedar Backup, your writer device must be properly configured. If you have configured your CD/DVD writer hardware to work through the normal filesystem device path, then you just need to know the path to the device on disk (something like `/dev/cdrw`). Cedar Backup will use the this device path both when talking to a command like **cdrecord** and when doing filesystem operations like running media validation.

Your other option is to configure your CD writer hardware like a SCSI device (either because it *is* a

SCSI device or because you are using some sort of interface that makes it look like one). In this case, Cedar Backup will use the SCSI id when talking to **cdrecord** and the device path when running filesystem operations.

See the section called "Configuring your Writer Device" for more information on writer devices and how they are configured.

> **Note**
>
> There is no need to set up your CD/DVD device if you have decided not to execute the store action.
>
> Due to the underlying utilities that Cedar Backup uses, the SCSI id may only be used for CD writers, *not* DVD writers.

Step 4: Configure your backup user.

Choose a user to be used for backups. Some platforms may come with a "ready made" backup user. For other platforms, 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 5: 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 filesystem 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/
```

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 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 (below) 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 confidental 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/DVD 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/DVD 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/DVD 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. [4] *To be safe, always enable the consistency check option in the store configuration section.*

Step 9: Modify the backup cron jobs.

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

For general information about using cron, see the manpage for crontab(5).

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.

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.

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.

---

[4] See "SF Bug Tracking" at http://cedar-backup.sourceforge.net/.

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

Note

See Appendix D, *Securing Password-less SSH Connections* for some important notes on how to optionally further secure password-less SSH connections to your clients.

Step 1: Decide when you will run your backup.

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 set of cron jobs. Although you won't create your cron jobs just yet, you should decide now when you will run your backup so you are prepared for later.

Backing up large directories and creating ISO filesystem 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.

Warning

Because of the way Cedar Backup works, you must ensure that your backup *always* runs on the first day of your configured week. This is because Cedar Backup will only clear incremental backup information and re-initialize your media when running on the first day of the week. If you skip running Cedar Backup on the first day of the week, your backups will likely be "confused" until the next week begins, or until you re-run the backup using the `--full` flag.

Step 2: 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 3: 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 AAAAB3NzaC1yc2EAAAABIwAAAIEA0vOKjlfwohPg1oPRdrmwHk75l3mI9Tb/WRZfVnu2Pw69
uyphM9wBLRo6QfOC2T8vZCB8o/ZIgtAM3tkM0UgQHxKBXAZ+H36TOgg7BcI20I93iGtzpsMA/uXQy8kH
HgZooYqQ9pw+ZduXgmPcAAv2b5eTm07wRqFt/U84k6bhTzs= user@machine
```

Step 4: Configure your backup user.

Choose a user to be used for backups. Some platforms may come with a "ready made" backup user. For other platforms, 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 5: 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 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 (below) 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 confidental 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.

Step 9: Modify the backup cron jobs.

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. [5]

Note

For general information about using cron, see the manpage for crontab(5).

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.

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

---

[5]See the section called "Coordination between Master and Clients" in Chapter 2, *Basic Concepts*.

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.

Tip

This setup procedure discusses how to set up Cedar Backup in the "normal case" for a master. If you would like to modify the way Cedar Backup works (for instance, by ignoring the store stage and just letting your backup sit in a staging directory), you can do that. You'll just have to modify the procedure below based on information in the remainder of the manual.

Step 1: Decide when you will run your backup.

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 set of cron jobs. Although you won't create your cron jobs just yet, you should decide now when you will run your backup so you are prepared for later.

Keep in mind that you do not necessarily have to run the collect action on the master. See notes further below for more information.

Backing up large directories and creating ISO filesystem 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.

Warning

Because of the way Cedar Backup works, you must ensure that your backup *always* runs on the first day of your configured week. This is because Cedar Backup will only clear incremental backup information and re-initialize your media when running on the first day of the week. If you skip running Cedar Backup on the first day of the week, your backups will likely be "confused" until the next week begins, or until you re-run the backup using the `--full` flag.

Step 2: 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 3: Configure your writer device.

Before using Cedar Backup, your writer device must be properly configured. If you have configured your CD/DVD writer hardware to work through the normal filesystem device path, then you just need to know the path to the device on disk (something like `/dev/cdrw`). Cedar Backup will use the this device path both when talking to a command like **cdrecord** and when doing filesystem operations like running media validation.

Your other option is to configure your CD writer hardware like a SCSI device (either because it *is* a SCSI device or because you are using some sort of interface that makes it look like one). In this case, Cedar Backup will use the SCSI id when talking to **cdrecord** and the device path when running filesystem operations.

See the section called "Configuring your Writer Device" for more information on writer devices and how they are configured.

> **Note**
>
> There is no need to set up your CD/DVD device if you have decided not to execute the store action.
>
> Due to the underlying utilities that Cedar Backup uses, the SCSI id may only be used for CD writers, *not* DVD writers.

Step 4: Configure your backup user.

Choose a user to be used for backups. Some platforms may come with a "ready made" backup user. For other platforms, 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 5: 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 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 would typically configure all four action-specific sections: collect, stage, store and purge.

Note

Note that the master can treat itself as a "client" peer for certain actions. As an example, if you run the collect action on the master, then you will stage that data by configuring a local peer representing the master.

Something else to keep in mind is that you do not really have to run the collect action on the master. For instance, you may prefer to just use your master machine as a "consolidation point" machine that just collects data from the other client machines in a backup pool. In that case, there is no need to collect data on the master itself.

The usual location for the Cedar Backup config file is `/etc/cback.conf`. If you change the location, make sure you edit your cronjobs (below) 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 confidental 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/DVD 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 to each client 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/DVD 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/DVD 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. [4] *To be safe, always enable the consistency check option in the store configuration section.*

Step 10: Modify the backup cron jobs.

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. [5]

Note

For general information about using cron, see the manpage for crontab(5).

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.

Configuring your Writer Device

Device Types

In order to execute the store action, you need to know how to identify your writer device. Cedar Backup supports two kinds of device types: CD writers and DVD writers. DVD writers are always referenced through a filesystem device name (i.e. `/dev/dvd`). CD writers can be referenced either through a SCSI id, or through a filesystem device name. Which you use depends on your operating system and hardware.

Devices identified by by device name

For all DVD writers, and for CD writers on certain platforms, you will configure your writer device using only a device name. If your writer device works this way, you should just specify <target_device> in configuration. You can either leave <target_scsi_id> blank or remove it completely. The writer device will be used both to write to the device and for filesystem operations — for instance, when the media needs to be mounted to run the consistency check.

Devices identified by SCSI id

Cedar Backup can use devices identified by SCSI id only when configured to use the `cdwriter` device type.

In order to use a SCSI device with Cedar Backup, you must know both the SCSI id <target_scsi_id> and the device name <target_device>. The SCSI id will be used to write to media using **cdrecord**; and the device name will be used for other filesystem operations.

A true SCSI device will always have an address `scsibus,target,lun` (i.e. `1,6,2`). This should hold true on most UNIX-like systems including Linux and the various BSDs (although I do not have a BSD system to test with currently). The SCSI address represents the location of your writer device on the one or more SCSI buses that you have available on your system.

On some platforms, it is possible to reference non-SCSI writer devices (i.e. an IDE CD writer) using an emulated SCSI id. If you have configured your non-SCSI writer device to have an emulated SCSI id, provide the filesystem device path in <target_device> and the SCSI id in <target_scsi_id>, just like for a real SCSI device.

You should note that in some cases, an emulated SCSI id takes the same form as a normal SCSI id, while in other cases you might see a method name prepended to the normal SCSI id (i.e. "ATA:1,1,1").

Linux Notes

On a Linux system, IDE writer devices often have a emulated SCSI address, which allows SCSI-based software to access the device through an IDE-to-SCSI interface. Under these circumstances, the first IDE writer device typically has an address `0,0,0`. However, support for the IDE-to-SCSI interface has been deprecated and is not well-supported in newer kernels (kernel 2.6.x and later).

Newer Linux kernels can address *ATA* or *ATAPI* drives without SCSI emulation by prepending a "method" indicator to the emulated device address. For instance, `ATA:0,0,0` or `ATAPI:0,0,0` are typical values.

However, even this interface is deprecated as of late 2006, so with relatively new kernels you may be better off using the filesystem device path directly rather than relying on any SCSI emulation.

Finding your Linux CD Writer

Here are some hints about how to find your Linux CD writer hardware. First, try to reference your device using the filesystem device path:

```
cdrecord -prcap dev=/dev/cdrom
```

Running this command on my hardware gives output that looks like this (just the top few lines):

```
Device type     : Removable CD-ROM
Version         : 0
Response Format: 2
Capabilities    :
Vendor_info     : 'LITE-ON '
Identification : 'DVDRW SOHW-1673S'
Revision        : 'JS02'
Device seems to be: Generic mmc2 DVD-R/DVD-RW.

Drive capabilities, per MMC-3 page 2A:
```

If this works, and the identifying information at the top of the output looks like your CD writer device, you've probably found a working configuration. Place the device path into <target_device> and leave <target_scsi_id> blank.

If this doesn't work, you should try to find an ATA or ATAPI device:

```
cdrecord -scanbus dev=ATA
cdrecord -scanbus dev=ATAPI
```

On my development system, I get a result that looks something like this for ATA:

```
scsibus1:
        1,0,0   100) 'LITE-ON ' 'DVDRW SOHW-1673S' 'JS02' Removable CD-ROM
        1,1,0   101) *
        1,2,0   102) *
        1,3,0   103) *
        1,4,0   104) *
        1,5,0   105) *
        1,6,0   106) *
        1,7,0   107) *
```

Again, if you get a result that you recognize, you have again probably found a working configuraton. Place the associated device path (in my case, /dev/cdrom) into <target_device> and put the emulated SCSI id (in this case, ATA:1,0,0) into <target_scsi_id>.

Any further discussion of how to configure your CD writer hardware is outside the scope of this document. If you have tried the hints above and still can't get things working, you may want to reference the *Linux CDROM HOWTO* (http://www.tldp.org/HOWTO/CDROM-HOWTO) or the *ATA RAID HOWTO* (http://www.tldp.org/HOWTO/ATA-RAID-HOWTO/index.html) for more information.

Mac OS X Notes

On a Mac OS X (darwin) system, things get strange. Apple has abandoned traditional SCSI device identifiers in favor of a system-wide resource id. So, on a Mac, your writer device will have a name something like IOCompactDiscServices (for a CD writer) or IODVDServices (for a DVD

writer). If you have multiple drives, the second drive probably has a number appended, i.e. `IODVDServices/2` for the second DVD writer. You can try to figure out what the name of your device is by grepping through the output of the command **ioreg -l**.[6]

Unfortunately, even if you can figure out what device to use, I can't really support the store action on this platform. In OS X, the "automount" function of the Finder interferes significantly with Cedar Backup's ability to mount and unmount media and write to the CD or DVD hardware. The Cedar Backup writer and image functionality does work on this platform, but the effort required to fight the operating system about who owns the media and the device makes it nearly impossible to execute the store action successfully.

If you are interested in some of my notes about what works and what doesn't on this platform, check out the documentation in the `doc/osx` directory in the source distribution.

Optimized Blanking Stategy

When the optimized blanking strategy has not been configured, Cedar Backup uses a simplistic approach: rewritable media is blanked at the beginning of every week, period.

Since rewritable media can be blanked only a finite number of times before becoming unusable, some users — especially users of rewritable DVD media with its large capacity — may prefer to blank the media less often.

If the optimized blanking strategy is configured, Cedar Backup will use a blanking factor and attempt to determine whether future backups will fit on the current media. If it looks like backups will fit, then the media will not be blanked.

This feature will only be useful (assuming single disc is used for the whole week's backups) if the estimated total size of the weekly backup is considerably smaller than the capacity of the media (no more than 50% of the total media capacity), and only if the size of the backup can be expected to remain fairly constant over time (no frequent rapid growth expected).

There are two blanking modes: daily and weekly. If the weekly blanking mode is set, Cedar Backup will only estimate future capacity (and potentially blank the disc) once per week, on the starting day of the week. If the daily blanking mode is set, Cedar Backup will estimate future capacity (and potentially blank the disc) every time it is run. *You should only use the daily blanking mode in conjunction with daily collect configuration, otherwise you will risk losing data.*

If you are using the daily blanking mode, you can typically set the blanking value to 1.0. This will cause Cedar Backup to blank the media whenever there is not enough space to store the current day's backup.

If you are using the weekly blanking mode, then finding the correct blanking factor will require some experimentation. Cedar Backup estimates future capacity based on the configured blanking factor. The disc will be blanked if the following relationship is true:

```
bytes available / (1 + bytes required) # blanking factor
```

Another way to look at this is to consider the blanking factor as a sort of (upper) backup growth

---

[6]Thanks to the file README.macosX in the cdrtools-2.01+01a01 source tree for this information

estimate:

```
Total size of weekly backup / Full backup size at the start of the week
```

This ratio can be estimated using a week or two of previous backups. For instance, take this example, where March 10 is the start of the week and March 4 through March 9 represent the incremental backups from the previous week:

```
/opt/backup/staging# du -s 2007/03/*
3040     2007/03/01
3044     2007/03/02
6812     2007/03/03
3044     2007/03/04
3152     2007/03/05
3056     2007/03/06
3060     2007/03/07
3056     2007/03/08
4776     2007/03/09
6812     2007/03/10
11824    2007/03/11
```

In this case, the ratio is approximately 4:

```
6812 + (3044 + 3152 + 3056 + 3060 + 3056 + 4776) / 6812 = 3.9571
```

To be safe, you might choose to configure a factor of 5.0.

Setting a higher value reduces the risk of exceeding media capacity mid-week but might result in blanking the media more often than is necessary.

If you run out of space mid-week, then the solution is to run the rebuild action. If this happens frequently, a higher blanking factor value should be used.

Chapter 6. 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>99</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 Subversion [1] 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.

Each configured 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**.

There are two different kinds of Subversion repositories at this writing: BDB (Berkeley Database) and FSFS (a "filesystem within a filesystem"). This extension backs up both kinds of repositories in the

---

[1] See http://subversion.org

same way, using **svnadmin dump** in an incremental mode.

It turns out that FSFS repositories can also be backed up just like any other filesystem directory. If you would rather do the backup that way, then use the normal collect action rather than this extension. If you decide to do that, be sure to consult the Subversion documentation and make sure you understand the limitations of this kind of backup. [2]

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>99</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>bzip2</compress_mode>
    <repository>
        <abs_path>/opt/public/svn/docs</abs_path>
    </repository>
    <repository>
        <abs_path>/opt/public/svn/web</abs_path>
        <compress_mode>gzip</compress_mode>
    </repository>
    <repository_dir>
        <abs_path>/opt/private/svn</abs_path>
        <collect_mode>daily</collect_mode>
    </repository_dir>
</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

---

[2]For instance, see the "Backups" section on this page: http://freehackers.org/~shlomif/svn-raweb-light/subversion.cgi/trunk/notes/fsfs

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 suppport 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 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 or repository directory must be configured.

    The `repository` subsection contains the following fields:

    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`.

    abs_path
    Absolute path of the Subversion repository to back up.

    Restrictions: Must be an absolute path.

repository_dir
    A Subversion parent repository directory be collected.

    This is a subsection which contains information about a Subversion parent repository directory to be

backed up. Any subdirectory immediately within this directory is assumed to be a Subversion repository, and will be backed up.

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

The `repository_dir` subsection contains the following fields:

`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`.

`abs_path`
Absolute path of the Subversion repository to back up.

Restrictions: Must be an absolute path.

`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 subversion parent 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:

`rel_path`
A relative path to be excluded from the backup.

The path is assumed to be relative to the subversion parent directory itself. For instance, if the configured subversion parent directory is `/opt/svn` a configured relative path of `software` would exclude the path `/opt/svn/software`.

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 `$`).

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

Restrictions: Must be non-empty

MySQL Extension

The MySQL Extension is a Cedar Backup extension used to back up MySQL [3] databases via the Cedar Backup command line. It is intended to be run either immediately before or immediately after the standard collect action.

Note

This extension always produces a full backup. There is currently no facility for making incremental backups. If/when someone has a need for this and can describe how to do it, I will update this extension or provide another.

The backup is done via the **mysqldump** command included with the MySQL product. Output can be compressed using **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 configured databases can be backed up by a single user. Often, the "root" database user will be used. An alternative is to create a separate MySQL "backup" user and grant that user rights to read (but not write) various databases as needed. This second option is probably your best choice.

Warning

The extension accepts a username and password in configuration. However, you probably do not want to list those values in Cedar Backup configuration. This is because Cedar Backup will provide these values to **mysqldump** via the command-line `--user` and `--password` switches, which will be visible to other users in the process listing.

Instead, you should configure the username and password in one of MySQL's configuration files. Typically, that would be done by putting a stanza like this in `/root/.my.cnf`:

```
[mysqldump]
user     = root
password = <secret>
```

Of course, if you are executing the backup as a user other than root, then you would create the file in that user's home directory instead.

As a side note, it is also possible to configure `.my.cnf` such that Cedar Backup can back up a remote database server:

```
[mysqldump]
host = remote.host
```

---
[3]See http://www.mysql.com

> For this to work, you will also need to grant privileges properly for the user which is executing the backup. See your MySQL documentation for more information about how this can be done.
>
> Regardless of whether you are using `~/.my.cnf` or `/etc/cback.conf` to store database login and password information, you should be careful about who is allowed to view that information. Typically, this means locking down permissions so that only the file owner can read the file contents (i.e. use mode `0600`).

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>99</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>
    <compress_mode>bzip2</compress_mode>
    <all>Y</all>
</mysql>
```

If you have decided to configure login information in Cedar Backup rather than using MySQL configuration, then you would add the username and password fields to configuration:

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

The following elements are part of the MySQL configuration section:

user
    Database user.

73

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).

This value is optional. You should probably configure the username and password in MySQL configuration instead, as discussed above.

Restrictions: If provided, must be non-empty.

password
: Password associated with the database user.

This value is optional. You should probably configure the username and password in MySQL configuration instead, as discussed above.

Restrictions: If provided, 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.

PostgreSQL Extension

Community-contributed Extension

This is a community-contributed extension provided by Antoine Beaupre ("The Anarcat"). I have

added regression tests around the configuration parsing code and I will maintain this section in the user manual based on his source code documentation.

Unfortunately, I don't have any PostgreSQL databases with which to test the functional code. While I have code-reviewed the code and it looks both sensible and safe, I have to rely on the author to ensure that it works properly.

The PostgreSQL Extension is a Cedar Backup extension used to back up PostgreSQL [4] 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 **pg_dump** or **pg_dumpall** commands included with the PostgreSQL product. Output can be compressed using **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 the current user has passwordless access to the database since there is no easy way to pass a password to the **pg_dump** client. This can be accomplished using appropriate configuration in the **pg_hda.conf** file.

This extension always produces a full backup. There is currently no facility for making incremental backups.

> ## Warning
>
> Once you place PostgreSQL configuration into the Cedar Backup configuration file, you should be careful about who is allowed to see that information. This is because PostgreSQL configuration will contain information about available PostgreSQL databases and usernames. Typically, you might want to lock down permissions so that only the file owner can read the file contents (i.e. use mode `0600`).

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

```
<extensions>
   <action>
      <name>postgresql</name>
      <module>CedarBackup2.extend.postgresql</module>
      <function>executeAction</function>
      <index>99</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 `postgresql` configuration section. This is an example PostgreSQL configuration section:

```
<postgresql>
   <compress_mode>bzip2</compress_mode>
```

---

[4]See http://www.postgresql.org/

```
    <user>username</user>
    <all>Y</all>
</postgresql>
```

If you decide to back up specific databases, then you would list them individually, like this:

```
<postgresql>
    <compress_mode>bzip2</compress_mode>
    <user>username</user>
    <all>N</all>
    <database>db1</database>
    <database>db2</database>
</postgresql>
```

The following elements are part of the PostgreSQL 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.

    This value is optional.

    Consult your PostgreSQL documentation for information on how to configure a default database user outside of Cedar Backup, and for information on how to specify a database password when you configure a user within Cedar Backup. You will probably want to modify **pg_hda.conf**.

    Restrictions: If provided, must be non-empty.

compress_mode
    Compress mode.

    PostgreSQL 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 PostgreSQL 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.

Mbox Extension

The Mbox Extension is a Cedar Backup extension used to incrementally back up UNIX-style "mbox" mail folders via the Cedar Backup command line. It is intended to be run either immediately before or immediately after the standard collect action.

Mbox mail folders are not well-suited to being backed up by the normal Cedar Backup incremental backup process. This is because active folders are typically appended to on a daily basis. This forces the incremental backup process to back them up every day in order to avoid losing data. This can result in quite a bit of wasted space when backing up large mail folders.

What the mbox extension does is leverage the **grepmail** utility to back up only email messages which have been received since the last incremental backup. This way, even if a folder is added to every day, only the recently-added messages are backed up. This can potentially save a lot of space.

Each configured mbox file or directory 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**.

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

```
<extensions>
    <action>
        <name>mbox</name>
        <module>CedarBackup2.extend.mbox</module>
        <function>executeAction</function>
        <index>99</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 mbox configuration section. This is an example mbox configuration section:

```
<mbox>
    <collect_mode>incr</collect_mode>
    <compress_mode>gzip</compress_mode>
    <file>
        <abs_path>/home/user1/mail/greylist</abs_path>
        <collect_mode>daily</collect_mode>
    </file>
```

77

```
    <dir>
        <abs_path>/home/user2/mail</abs_path>
    </dir>
    <dir>
        <abs_path>/home/user3/mail</abs_path>
        <exclude>
            <rel_path>spam</rel_path>
            <pattern>.*debian.*</pattern>
        </exclude>
    </dir>
</mbox>
```

Configuration is much like the standard collect action. Differences come from the fact that mbox directories are *not* collected recursively.

Unlike collect configuration, exclusion information can only be configured at the mbox directory level (there are no global exclusions). Another difference is that no absolute exclusion paths are allowed — only relative path exclusions and patterns.

The following elements are part of the mbox configuration section:

collect_mode
    Default collect mode.

    The collect mode describes how frequently an mbox file or directory is backed up. The mbox 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 files or directories (below) may override this value. If *all* individual files or directories provide their own value, then this default value may be omitted from configuration.

    Note: if your backup device does not suppport 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.

    Mbox file or directory backups are just text, 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 files or directories (below) may override this value. If *all* individual files or directories provide their own value, then this default value may be omitted from configuration.

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

file
    An individual mbox file to be collected.

    This is a subsection which contains information about an individual mbox file to be backed up.

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

The file subsection contains the following fields:

`collect_mode`
Collect mode for this file.

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 file.

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`.

`abs_path`
Absolute path of the mbox file to back up.

Restrictions: Must be an absolute path.

`dir`
An mbox directory to be collected.

This is a subsection which contains information about an mbox directory to be backed up. An mbox directory is a directory containing mbox files. Every file in an mbox directory is assumed to be an mbox file. Mbox directories are *not* collected recursively. Only the files immediately within the configured directory will be backed-up and any subdirectories will be ignored.

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

The dir subsection contains the following fields:

`collect_mode`
Collect mode for this file.

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 file.

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`.

`abs_path`
Absolute path of the mbox directory to back up.

Restrictions: Must be an absolute path.

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 mbox 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:

rel_path
A relative path to be excluded from the backup.

The path is assumed to be relative to the mbox directory itself. For instance, if the configured mbox directory is /home/user2/mail a configured relative path of SPAM would exclude the path /home/user2/mail/SPAM.

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 $).

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

Restrictions: Must be non-empty

Encrypt Extension

The Encrypt Extension is a Cedar Backup extension used to encrypt backups. It does this by encrypting the contents of a master's staging directory each day after the stage action is run. This way, backed-up data is encrypted both when sitting on the master and when written to disc. This extension must be run before the standard store action, otherwise unencrypted data will be written to disc.

There are several differents ways encryption could have been built in to or layered on to Cedar Backup. I asked the mailing list for opinions on the subject in January 2007 and did not get a lot of feedback, so I chose the option that was simplest to understand and simplest to implement. If other encryption use cases make themselves known in the future, this extension can be enhanced or replaced.

Currently, this extension supports only GPG. However, it would be straightforward to support other public-key encryption mechanisms, such as OpenSSL.

Warning

If you decide to encrypt your backups, be *absolutely sure* that you have your GPG secret key saved off someplace safe — someplace other than on your backup disc. If you lose your secret key, your backup will be useless.

I suggest that before you rely on this extension, you should execute a dry run and make

sure you can successfully decrypt the backup that is written to disc.

Before configuring the Encrypt extension, you must configure GPG. Either create a new keypair or use an existing one. Determine which user will execute your backup (typically root) and have that user import *and lsign* the public half of the keypair. Then, save off the secret half of the keypair someplace safe, apart from your backup (i.e. on a floppy disk or USB drive). Make sure you know the recipient name associated with the public key because you'll need it to configure Cedar Backup. (If you can run **gpg -e -r "Recipient Name" file.txt** and it executes cleanly with no user interaction required, you should be OK.)

An encrypted backup has the same file structure as a normal backup, so all of the instructions in Appendix C, *Data Recovery* apply. The only difference is that encrypted files will have an additional `.gpg` extension (so for instance `file.tar.gz` becomes `file.tar.gz.gpg`). To recover decrypted data, simply log on as a user which has access to the secret key and decrypt the `.gpg` file that you are interested in. Then, recover the data as usual.

Note: I am being intentionally vague about how to configure and use GPG, because I do not want to encourage neophytes to blindly use this extension. If you do not already understand GPG well enough to follow the two paragraphs above, *do not use this extension*. Instead, before encrypting your backups, check out the excellent GNU Privacy Handbook at http://www.gnupg.org/gph/en/manual.html and gain an understanding of how encryption can help you or hurt you.

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

```
<extensions>
    <action>
        <name>encrypt</name>
        <module>CedarBackup2.extend.encrypt</module>
        <function>executeAction</function>
        <index>301</index>
    </action>
</extensions>
```

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

```
<encrypt>
    <encrypt_mode>gpg</encrypt_mode>
    <encrypt_target>Backup User</encrypt_target>
</encrypt>
```

The following elements are part of the Encrypt configuration section:

encrypt_mode
    Encryption mode.

    This value specifies which encryption mechanism will be used by the extension.

81

Currently, only the GPG public-key encryption mechanism is supported.

Restrictions: Must be `gpg`.

`encrypt_target`
Encryption target.

The value in this field is dependent on the encryption mode. For the `gpg` mode, this is the name of the recipient whose public key will be used to encrypt the backup data, i.e. the value accepted by **gpg -r**.

Split Extension

The Split Extension is a Cedar Backup extension used to split up large files within staging directories. It is probably only useful in combination with the **cback-span** command, which requires individual files within staging directories to each be smaller than a single disc.

You would normally run this action immediately after the standard stage action, but you could also choose to run it by hand immediately before running **cback-span**.

The split extension uses the standard UNIX **split** tool to split the large files up. This tool simply splits the files on bite-size boundaries. It has no knowledge of file formats.

Note: this means that in order to recover the data in your original large file, you must have every file that the original file was split into. Think carefully about whether this is what you want. It doesn't sound like a huge limitation. However, **cback-span** might put an indivdual file on *any* disc in a set — the files split from one larger file will not necessarily be together. That means you will probably need every disc in your backup set in order to recover any data from the backup set.

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

```
<extensions>
    <action>
        <name>split</name>
        <module>CedarBackup2.extend.split</module>
        <function>executeAction</function>
        <index>299</index>
    </action>
</extensions>
```

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

```
<split>
    <size_limit>250 MB</size_limit>
    <split_size>100 MB</split_size>
</split>
```

The following elements are part of the Split configuration section:

size_limit
    Size limit.

    Files with a size strictly larger than this limit will be split by the extension.

    You can enter this value in two different forms. It can either be a simple number, in which case the value is assumed to be in bytes; or it can be a number followed by a unit (KB, MB, GB).

    Valid examples are "10240", "250 MB" or "1.1 GB".

    Restrictions: Must be a size as described above.

split_size
    Split size.

    This is the size of the chunks that a large file will be split into. The final chunk may be smaller if the split size doesn't divide evenly into the file size.

    You can enter this value in two different forms. It can either be a simple number, in which case the value is assumed to be in bytes; or it can be a number followed by a unit (KB, MB, GB).

    Valid examples are "10240", "250 MB" or "1.1 GB".

    Restrictions: Must be a size as described above.

Capacity Extension

The capacity extension checks the current capacity of the media in the writer and prints a warning if the media exceeds an indicated capacity. The capacity is indicated either by a maximum percentage utilized or by a minimum number of bytes that must remain unused.

This action can be run at any time, but is probably best run as the last action on any given day, so you get as much notice as possible that your media is full and needs to be replaced.

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

```
<extensions> <action>
      <name>capacity</name>
      <module>CedarBackup2.extend.capacity</module>
      <function>executeAction</function>
      <index>299</index>
   </action>
</extensions>
```

This extension relies on the options and store configuration sections in the standard Cedar Backup configuration file, and then also requires its own capacity configuration section. This is an example Capacity configuration section that configures the extension to warn if the media is more than 95.5% full:

```
<capacity>
    <max_percentage>95.5</max_percentage>
</capacity>
```

This example configures the extension to warn if the media has fewer than 16 MB free:

```
<capacity>
    <min_bytes>16 MB</min_bytes>
</capacity>
```

The following elements are part of the Capacity configuration section:

`max_percentage`

Maximum percentage of the media that may be utilized.

You must provide either this value *or* the `min_bytes` value.

Restrictions: Must be a floating point number between 0.0 and 100.0

`min_bytes`

Minimum number of free bytes that must be available.

You can enter this value in two different forms. It can either be a simple number, in which case the value is assumed to be in bytes; or it can be a number followed by a unit (KB, MB, GB).

Valid examples are "10240", "250 MB" or "1.1 GB".

You must provide either this value *or* the `max_percentage` value.

Restrictions: Must be a byte quantity as described above.

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 succesfully 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>foo</module>
        <function>bar</function>
        <index>101</index>
    </action>
</extensions>
```

In this case, the action "database" has been mapped to the extension function `foo.bar()`.

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 with the _ character. Furthermore, extensions should only rely on parts of the public interface that are documented in the online Epydoc documentation.

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.

8. Extensions must be written to assume an empty locale set (no $LC_* settings) and $LANG=C. For the typical open-source software project, this would imply writing output-parsing code against the English localization (if any). The executeCommand function does sanitize the environment to enforce this configuration.

Extension functions take three arguments: the path to configuration on disk, a 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 and BSD distributions should include it (although Debian "woody" does not include it.)

Source	URL
upstream	http://www.python.org
Debian	http://packages.debian.org/stable/python/python2.3
Gentoo	http://packages.gentoo.org/packages/?category=dev-lang;name=python;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=python
Mac OS X (fink)	http://fink.sourceforge.net/pdb/package.php/python23

> 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/stable/net/ssh
Gentoo	http://packages.gentoo.org/packages/?category=net-misc;name=openssh;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=openssh
Mac OS X	*built-in*

> 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 filesystem images that can later be written to backup media.

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

Source	URL
Mac OS X (fink)	http://fink.sourceforge.net/pdb/package.php/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 CD media in a backup device.

Source	URL
upstream	http://freshmeat.net/projects/cdrecord/
Debian	http://packages.debian.org/stable/otherosfs/cdrecord
Gentoo	http://packages.gentoo.org/packages/?category=app-cdr;name=cdrtools;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=cdrecord
Mac OS X (fink)	http://fink.sourceforge.net/pdb/search.php?summary=cdrecord

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

dvd+rw-tools
The dvd+rw-tools package provides the **growisofs** utility, which is used to write ISO images to DVD media in a backup device.

Source	URL
upstream	http://fy.chalmers.se/~appro/linux/DVD+RW/
Debian	http://packages.debian.org/stable/utils/dvd+rw-tools
Gentoo	http://packages.gentoo.org/packages/?category=app-cdr;name=dvd%2Brw-tools
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=dvd+rw-tools
Mac OS X (fink)	http://pdb.finkproject.org/pdb/package.php/dvd+rw-tools

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

Source	URL
Debian	http://packages.debian.org/stable/utils/eject
Gentoo	http://packages.gentoo.org/packages/?category=sys-apps;name=eject;
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=eject
Mac OS X (fink)	http://fink.sourceforge.net/pdb/package.php/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/DVD 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/stable/base/mount
Gentoo	*unknown*
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=mount
Mac OS X	*built-in*

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?

I have classified Mac OS X "built-in" because that operating system does contain a mount command. However, it isn't really compatible with Cedar Backup's idea of mount, and in fact what Cedar Backup needs is closer to the **hdiutil** command. However, there are other issues related to that command, which is why the store action is not really supported on Mac OS X.

grepmail

The **grepmail** command is used by the mbox extension to pull out only recent messages from mbox mail folders.

Source	URL
upstream	http://freshmeat.net/projects/grepmail/
Debian	http://packages.debian.org/stable/mail/grepmail
Gentoo	http://packages.gentoo.org/packages/?category=net-mail;name=grepmail
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=grepmail
Mac OS X	http://pdb.finkproject.org/pdb/package.php/grepmail

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

gpg

The **gpg** command is used by the encrypt extension to encrypt files.

Source	URL
upstream	http://freshmeat.net/projects/gnupg/
Debian	http://packages.debian.org/stable/utils/gnupg
Gentoo	http://packages.gentoo.org/packages/?category=app-crypt;name=gnupg
RPM	http://rpmfind.net/linux/rpm2html/search.php?query=gnupg
Mac OS X	http://pdb.finkproject.org/pdb/package.php/gnupg

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

split

The **split** command is used by the split extension to split up large files.

This command is typically part of the core operating system install and is not distributed in a separate package.

Appendix C. Data Recovery

Finding your Data

The first step in data recovery is finding the data that you want to recover. You need to decide whether you are going to to restore off backup media, or out of some existing staging data that has not yet been purged. The only difference is, if you purge staging data less frequently than once per week, you might have some data available in the staging directories which would not be found on your backup media, depending on how you rotate your media. (And of course, if your system is trashed or stolen, you probably will not have access to your old staging data in any case.)

Regardless of the data source you choose, you will find the data organized in the same way. The remainder of these examples will work off an example backup disc, but the contents of the staging directory will look pretty much like the contents of the disc, with data organized first by date and then by backup peer name.

This is the root directory of my example disc:

```
root:/mnt/cdrw# ls -l
total 4
drwxr-x---  3 backup backup 4096 Sep 01 06:30 2005/
```

In this root directory is one subdirectory for each year represented in the backup. In this example, the backup represents data entirely from the year 2005. If your configured backup week happens to span a year boundary, there would be two subdirectories here (for example, one for 2005 and one for 2006).

Within each year directory is one subdirectory for each month represented in the backup.

```
root:/mnt/cdrw/2005# ls -l
total 2
dr-xr-xr-x  6 root root 2048 Sep 11 05:30 09/
```

In this example, the backup represents data entirely from the month of September, 2005. If your configured backup week happens to span a month boundary, there would be two subdirectories here (for example, one for August 2005 and one for September 2005).

Within each month directory is one subdirectory for each day represented in the backup.

```
root:/mnt/cdrw/2005/09# ls -l
total 8
dr-xr-xr-x  5 root root 2048 Sep  7 05:30 07/
dr-xr-xr-x  5 root root 2048 Sep  8 05:30 08/
dr-xr-xr-x  5 root root 2048 Sep  9 05:30 09/
dr-xr-xr-x  5 root root 2048 Sep 11 05:30 11/
```

91

Depending on how far into the week your backup media is from, you might have as few as one daily directory in here, or as many as seven.

Within each daily directory is a stage indicator (indicating when the directory was staged) and one directory for each peer configured in the backup:

```
root:/mnt/cdrw/2005/09/07# ls -l
total 10
dr-xr-xr-x  2 root root 2048 Sep  7 02:31 host1/
-r--r--r--  1 root root    0 Sep  7 03:27 cback.stage
dr-xr-xr-x  2 root root 4096 Sep  7 02:30 host2/
dr-xr-xr-x  2 root root 4096 Sep  7 03:23 host3/
```

In this case, you can see that my backup includes three machines, and that the backup data was staged on September 7, 2005 at 03:27.

Within the directory for a given host are all of the files collected on that host. This might just include tarfiles from a normal Cedar Backup collect run, and might also include files "collected" from Cedar Backup extensions or by other third-party processes on your system.

```
root:/mnt/cdrw/2005/09/07/host1# ls -l
total 157976
-r--r--r--  1 root root 11206159 Sep  7 02:30 boot.tar.bz2
-r--r--r--  1 root root        0 Sep  7 02:30 cback.collect
-r--r--r--  1 root root     3199 Sep  7 02:30 dpkg-selections.txt.bz2
-r--r--r--  1 root root   908325 Sep  7 02:30 etc.tar.bz2
-r--r--r--  1 root root      389 Sep  7 02:30 fdisk-l.txt.bz2
-r--r--r--  1 root root  1003100 Sep  7 02:30 ls-laR.txt.bz2
-r--r--r--  1 root root    19800 Sep  7 02:30 mysqldump.txt.bz2
-r--r--r--  1 root root  4133372 Sep  7 02:30 opt-local.tar.bz2
-r--r--r--  1 root root 44794124 Sep  8 23:34 opt-public.tar.bz2
-r--r--r--  1 root root 30028057 Sep  7 02:30 root.tar.bz2
-r--r--r--  1 root root  4747070 Sep  7 02:30 svndump-0:782-opt-svn-repo1.txt.bz2
-r--r--r--  1 root root   603863 Sep  7 02:30 svndump-0:136-opt-svn-repo2.txt.bz2
-r--r--r--  1 root root   113484 Sep  7 02:30 var-lib-jspwiki.tar.bz2
-r--r--r--  1 root root 19556660 Sep  7 02:30 var-log.tar.bz2
-r--r--r--  1 root root 14753855 Sep  7 02:30 var-mail.tar.bz2
```

As you can see, I back up variety of different things on host1. I run the normal collect action, as well as the sysinfo, mysql and subversion extensions. The resulting backup files are named in a way that makes it easy to determine what they represent.

Files of the form `*.tar.bz2` represent directories backed up by the collect action. The first part of the name (before ".tar.bz2"), represents the path to the directory. For example, `boot.tar.gz` contains data from `/boot`, and `var-lib-jspwiki.tar.bz2` contains data from `/var/lib/jspwiki`.

The `fdisk-l.txt.bz2`, `ls-laR.tar.bz2` and `dpkg-selections.tar.bz2` files are produced by the sysinfo extension.

The `mysqldump.txt.bz2` file is produced by the mysql extension. It represents a system-wide database dump, because I use the "all" flag in configuration. If I were to configure Cedar Backup to

dump individual datbases, then the filename would contain the database name (something like `mysqldump-bugs.txt.bz2`).

Finally, the files of the form `svndump-*.txt.bz2` are produced by the subversion extension. There is one dump file for each configured repository, and the dump file name represents the name of the repository and the revisions in that dump. So, the file `svndump-0:782-opt-svn-repo1.txt.bz2` represents revisions 0-782 of the repository at `/opt/svn/repo1`. You can tell that this file contains a full backup of the repository to this point, because the starting revision is zero. Later incremental backups would have a non-zero starting revision, i.e. perhaps 783-785, followed by 786-800, etc.

Recovering Filesystem Data

Filesystem data is gathered by the standard Cedar Backup collect action. This data is placed into files of the form `*.tar`. The first part of the name (before ".tar"), represents the path to the directory. For example, `boot.tar` would contain data from `/boot`, and `var-lib-jspwiki.tar` would contain data from `/var/lib/jspwiki`. (As a special case, data from the root directory would be placed in `-.tar`). Remember that your tarfile might have a bzip2 (`.bz2`) or gzip (`.gz`) extension, depending on what compression you specified in configuration.

If you are using full backups every day, the latest backup data is always within the latest daily directory stored on your backup media or within your staging directory. If you have some or all of your directories configured to do incremental backups, then the first day of the week holds the full backups and the other days represent incremental differences relative to that first day of the week.

> **Where to extract your backup**
>
> If you are restoring a home directory or some other non-system directory as part of a full restore, it is probably fine to extract the backup directly into the filesystem.
>
> If you are restoring a system directory like `/etc` as part of a full restore, extracting directly into the filesystem is likely to break things, especially if you re-installed a newer version of your operating system than the one you originally backed up. It's better to extract directories like this to a temporary location and pick out only the files you find you need.
>
> When doing a partial restore, I suggest *always* extracting to a temporary location. Doing it this way gives you more control over what you restore, and helps you avoid compounding your original problem with another one (like overwriting the wrong file, oops).

Full Restore

To do a full system restore, find the newest applicable full backup and extract it. If you have some incremental backups, extract them into the same place as the full backup, one by one starting from oldest to newest. (This way, if a file changed every day you will always get the latest one.)

All of the backed-up files are stored in the tar file in a relative fashion, so you can extract from the tar file either directly into the filesystem, or into a temporary location.

For example, to restore `boot.tar.bz2` directly into `/boot`, execute **tar** from your root directory (`/`):

```
root:/# bzcat boot.tar.bz2 | tar xvf -
```

Of course, use **zcat** or just **cat**, depending on what kind of compression is in use.

If you want to extract `boot.tar.gz` into a temporary location like `/tmp/boot` instead, just change directories first. In this case, you'd execute the **tar** command from within `/tmp` instead of `/`.

```
root:/tmp# bzcat boot.tar.bz2 | tar xvf -
```

Again, use **zcat** or just **cat** as appropriate.

For more information, you might want to check out the manpage or GNU info documentation for the **tar** command.

Partial Restore

Most users will need to do a partial restore much more frequently than a full restore. Perhaps you accidentally removed your home directory, or forgot to check in some version of a file before deleting it. Or, perhaps the person who packaged Apache for your system blew away your web server configuration on upgrade (it happens). The solution to these and other kinds of problems is a partial restore (assuming you've backed up the proper things).

The procedure is similar to a full restore. The specific steps depend on how much information you have about the file you are looking for. Where with a full restore, you can confidently extract the full backup followed by each of the incremental backups, this might not be what you want when doing a partial restore. You may need to take more care in finding the right version of a file — since the same file, if changed frequently, would appear in more than one backup.

Start by finding the backup media that contains the file you are looking for. If you rotate your backup media, and your last known "contact" with the file was a while ago, you may need to look on older media to find it. This may take some effort if you are not sure when the change you are trying to correct took place.

Once you have decided to look at a particular piece of backup media, find the correct peer (host), and look for the file in the full backup:

```
root:/tmp# bzcat boot.tar.bz2 | tar tvf - path/to/file
```

Of course, use **zcat** or just **cat**, depending on what kind of compression is in use.

The `tvf` tells **tar** to search for the file in question and just list the results rather than extracting the file. Note that the filename is relative (with no starting `/`). Alternately, you can omit the `path/to/file` and search through the output using **more** or **less**

If you haven't found what you are looking for, work your way through the incremental files for the directory in question. One of them may also have the file if it changed during the course of the backup. Or, move to older or newer media and see if you can find the file there.

Once you have found your file, extract it using xvf:

```
root:/tmp# bzcat boot.tar.bz2 | tar xvf - path/to/file
```

Again, use **zcat** or just **cat** as appropriate.

Inspect the file and make sure it's what you're looking for. Again, you may need to move to older or newer media to find the exact version of your file.

For more information, you might want to check out the manpage or GNU info documentation for the **tar** command.

Recovering MySQL Data

MySQL data is gathered by the Cedar Backup mysql extension. This extension always creates a full backup each time it runs. This wastes some space, but makes it easy to restore database data. The following procedure describes how to restore your MySQL database from the backup.

> ### Warning
>
> I am not a MySQL expert. I am providing this information for reference. I have tested these procedures on my own MySQL installation; however, I only have a single database for use by Bugzilla, and I may have misunderstood something with regard to restoring individual databases as a user other than root. If you have any doubts, test the procedure below before relying on it!
>
> MySQL experts and/or knowledgable Cedar Backup users: feel free to write me and correct any part of this procedure.

First, find the backup you are interested in. If you have specified "all databases" in configuration, you will have a single backup file, called mysqldump.txt. If you have specified individual databases in configuration, then you will have files with names like mysqldump-database.txt instead. In either case, your file might have a .gz or .bz2 extension depending on what kind of compression you specified in configuration.

If you are restoring an "all databases" backup, make sure that you have correctly created the root user and know its password. Then, execute:

```
daystrom:/# bzcat mysqldump.txt.bz2 | mysql -p -u root
```

Of course, use **zcat** or just **cat**, depending on what kind of compression is in use.

Because the database backup includes CREATE DATABASE SQL statements, this command should take care of creating all of the databases within the backup, as well as populating them.

If you are restoring a backup for a specific database, you have two choices. If you have a root login, you can use the same command as above:

```
daystrom:/# bzcat mysqldump-database.txt.bz2 | mysql -p -u root
```

Otherwise, you can create the database and its login first (or have someone create it) and then use a database-specific login to execute the restore:

```
daystrom:/# bzcat mysqldump-database.txt.bz2 | mysql -p -u user database
```

Again, use **zcat** or just **cat** as appropriate.

For more information on using MySQL, see the documentation on the MySQL web site, http://mysql.org/, or the manpages for the **mysql** and **mysqldump** commands.

Recovering Subversion Data

Subversion data is gathered by the Cedar Backup subversion extension. Cedar Backup will create either full or incremental backups, but the procedure for restoring is the same for both. Subversion backups are always taken on a per-repository basis. If you need to restore more than one repository, follow the procedures below for each repository you are interested in.

First, find the backup or backups you are interested in. Typically, you will need the full backup from the first day of the week and each incremental backup from the other days of the week.

The subversion extension creates files of the form `svndump-*.txt`. These files might have a `.gz` or `.bz2` extension depending on what kind of compression you specified in configuration. There is one dump file for each configured repository, and the dump file name represents the name of the repository and the revisions in that dump. So, the file `svndump-0:782-opt-svn-repo1.txt.bz2` represents revisions 0-782 of the repository at `/opt/svn/repo1`. You can tell that this file contains a full backup of the repository to this point, because the starting revision is zero. Later incremental backups would have a non-zero starting revision, i.e. perhaps 783-785, followed by 786-800, etc.

Next, if you still have the old Subversion repository around, you might want to just move it off (rename the top-level directory) before executing the restore. Or, you can restore into a temporary directory and rename it later to its real name once you've checked it out. That is what my example below will show.

Next, you need to create a new Subversion repository to hold the restored data. This example shows an FSFS repository, but that is an arbitrary choice. You can restore from an FSFS backup into a FSFS repository or a BDB repository. The Subversion dump format is "backend-agnostic".

```
root:/tmp# svnadmin create --fs-type=fsfs testrepo
```

Next, load the full backup into the repository:

```
root:/tmp# bzcat svndump-0:782-opt-svn-repo1.txt.bz2 | svnadmin load testrepo
```

Of course, use **zcat** or just **cat**, depending on what kind of compression is in use.

Follow that with loads for each of the incremental backups:

```
root:/tmp# bzcat svndump-783:785-opt-svn-repo1.txt.bz2 │ svnadmin load testrepo
root:/tmp# bzcat svndump-786:800-opt-svn-repo1.txt.bz2 │ svnadmin load testrepo
```

Again, use **zcat** or just **cat** as appropriate.

When this is done, your repository will be restored to the point of the last commit indicated in the svndump file (in this case, to revision 800).

Note

Note: don't be surprised if, when you test this, the restored directory doesn't have exactly the same contents as the original directory. I can't explain why this happens, but if you execute **svnadmin dump** on both old and new repositories, the results are identical. This means that the repositories do contain the same content.

For more information on using Subversion, see the book *Version Control with Subversion* (http://svnbook.red-bean.com/) or the *Subversion FAQ* (http://subversion.tigris.org/faq.html).

Recovering Mailbox Data

Mailbox data is gathered by the Cedar Backup mbox extension. Cedar Backup will create either full or incremental backups, but both kinds of backups are treated identically when restoring.

Individual mbox files and mbox directories are treated a little differently, since individual files are just compressed, but directories are collected into a tar archive.

First, find the backup or backups you are interested in. Typically, you will need the full backup from the first day of the week and each incremental backup from the other days of the week.

The mbox extension creates files of the form `mbox-*`. Backup files for individual mbox files might have a `.gz` or `.bz2` extension depending on what kind of compression you specified in configuration. Backup files for mbox directories will have a `.tar`, `.tar.gz` or `.tar.bz2` extension, again depending on what kind of compression you specified in configuration.

There is one backup file for each configured mbox file or directory. The backup file name represents the name of the file or directory and the date it was backed up. So, the file `mbox-20060624-home-user-mail-greylist` represents the backup for `/home/user/mail/greylist` run on 24 Jun 2006. Likewise, `mbox-20060624-home-user-mail.tar` represents the backup for the `/home/user/mail` directory run on that same date.

Once you have found the files you are looking for, the restoration procedure is fairly simple. First, concatenate all of the backup files together. Then, use grepmail to eliminate duplicate messages (if any).

Here is an example for a single backed-up file:

```
root:/tmp# rm restore.mbox # make sure it's not left over
root:/tmp# cat mbox-20060624-home-user-mail-greylist >> restore.mbox
root:/tmp# cat mbox-20060625-home-user-mail-greylist >> restore.mbox
root:/tmp# cat mbox-20060626-home-user-mail-greylist >> restore.mbox
root:/tmp# grepmail -a -u restore.mbox > nodups.mbox
```

At this point, `nodups.mbox` contains all of the backed-up messages from `/home/user/mail/greylist`.

Of course, if your backups are compressed, you'll have to use **zcat** or **bzcat** rather than just **cat**.

If you are backing up mbox directories rather than individual files, see the filesystem instructions for notes on now to extract the individual files from inside tar archives. Extract the files you are interested in, and then concatenate them together just like shown above for the individual case.

Recovering Data split by the Split Extension

The Split extension takes large files and splits them up into smaller files. Typically, it would be used in conjunction with the **cback-span** command.

The split up files are not difficult to work with. Simply find all of the files — which could be split between multiple discs — and concatenate them together.

```
root:/tmp# rm usr-src-software.tar.gz  # make sure it's not there
root:/tmp# cat usr-src-software.tar.gz_00001 >> usr-src-software.tar.gz
root:/tmp# cat usr-src-software.tar.gz_00002 >> usr-src-software.tar.gz
root:/tmp# cat usr-src-software.tar.gz_00003 >> usr-src-software.tar.gz
```

Then, use the resulting file like usual.

Remember, you need to have *all* of the files that the original large file was split into before this will work. If you are missing a file, the result of the concatenation step will be either a corrupt file or a truncated file (depending on which chunks you did not include).

Appendix D. Securing Password-less SSH Connections

Cedar Backup relies on password-less public key SSH connections to make various parts of its backup process work. Password-less **scp** is used to stage files from remote clients to the master, and password-less **ssh** is used to execute actions on managed clients.

Normally, it is a good idea to avoid password-less SSH connections in favor of using an SSH agent. The SSH agent manages your SSH connections so that you don't need to type your passphrase over and over. You get most of the benefits of a password-less connection without the risk. Unfortunately, because Cedar Backup has to execute without human involvement (through a cron job), use of an agent really isn't feasable. We have to rely on true password-less public keys to give the master access to the client peers.

Traditionally, Cedar Backup has relied on a "segmenting" strategy to minimize the risk. Although the backup typically runs as root — so that all parts of the filesystem can be backed up — we don't use the root user for network connections. Instead, we use a dedicated backup user on the master to initiate network connections, and dedicated users on each of the remote peers to accept network connections.

With this strategy in place, an attacker with access to the backup user on the master (or even root access, really) can at best only get access to the backup user on the remote peers. We still concede a local attack vector, but at least that vector is restricted to an unprivileged user.

Some Cedar Backup users may not be comfortable with this risk, and others may not be able to implement the segmentation strategy — they simply may not have a way to create a login which is only used for backups.

So, what are these users to do? Fortunately there is a solution. The SSH authorized keys file supports a way to put a "filter" in place on an SSH connection. This excerpt is from the AUTHORIZED_KEYS FILE FORMAT section of man 8 sshd:

```
command="command"
    Specifies that the command is executed whenever this key is used for
    authentication.  The command supplied by the user (if any) is ignored.  The
    command is run on a pty if the client requests a pty; otherwise it is run
    without a tty.  If an 8-bit clean channel is required, one must not request
    a pty or should specify no-pty.  A quote may be included in the command by
    quoting it with a backslash.  This option might be useful to restrict
    certain public keys to perform just a specific operation.  An example might
    be a key that permits remote backups but nothing else.  Note that the client
    may specify TCP and/or X11 forwarding unless they are explicitly prohibited.
    Note that this option applies to shell, command or subsystem execution.
```

Essentially, this gives us a way to authenticate the commands that are being executed. We can either accept or reject commands, and we can even provide a readable error message for commands we reject. The filter is applied on the remote peer, to the key that provides the master access to the remote peer.

So, let's imagine that we have two hosts: master "mickey", and peer "minnie". Here is the original `~/.ssh/authorized_keys` file for the backup user on minnie (remember, this is all on one line in the file):

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxw7EnqVULBFgPcut3WYp3MsSpVB9q9iZ+awek120391k;m
=m=askdalkS82mlF7SusBTcXiCk1BGsg7axZ2sclgK+FfWV1Jm0/I9yo9FtAZ9U+MmpL901231asdkl;ai
1-2341=-a0sd=-sa0=1z= backup@mickey
```

This line is the public key that minnie can use to identify the backup user on mickey. Assuming that there is no passphrase on the private key back on mickey, the backup user on mickey can get direct access to minnie.

To put the filter in place, we add a command option to the key, like this:

```
command="/opt/backup/validate-backup" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxw7EnqVU
3MsSpVB9q9iZ+awek120391k;mm0c221=3=km=m=askdalkS82mlF7SusBTcXiCk1BGsg7axZ2sclgK+Ff
tAZ9U+MmpL901231asdkl;ai1-923ma9s=9=1-2341=-a0sd=-sa0=1z= backup@mickey
```

Basically, the command option says that whenever this key is used to successfully initiate a connection, the **/opt/backup/validate-backup** command will be run *instead of* the real command that came over the SSH connection. Fortunately, the interface gives the command access to certain shell variables that can be used to invoke the original command if you want to.

A very basic **validate-backup** script might look something like this:

```
#!/bin/bash
if [[ "${SSH_ORIGINAL_COMMAND}" == "ls -l" ]] ; then
    ${SSH_ORIGINAL_COMMAND}
else
   echo "Security policy does not allow command [${SSH_ORIGINAL_COMMAND}]."
   exit 1
fi
```

This script allows exactly **ls -l** and nothing else. If the user attempts some other command, they get a nice error message telling them that their command has been disallowed.

For remote commands executed over **ssh**, the original command is exactly what the caller attempted to invoke. For remote copies, the commands are either **scp -f file** (copy *from* the peer to the master) or **scp -t file** (copy *to* the peer from the master).

If you want, you can see what command SSH thinks it is executing by using **ssh -v** or **scp -v**. The command will be right at the top, something like this:

```
Executing: program /usr/bin/ssh host mickey, user (unspecified), command scp -v -f
OpenSSH_4.3p2 Debian-9, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /home/backup/.ssh/config
debug1: Applying options for daystrom
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug2: ssh_connect: needpriv 0
```

Omit the **-v** and you have your command: **scp -f .profile**.

For a normal, non-managed setup, you need to allow the following commands, where `/path/to/collect/` is replaced with the real path to the collect directory on the remote peer:

```
scp -f /path/to/collect/cback.collect
scp -f /path/to/collect/*
scp -t /path/to/collect/cback.stage
```

If you are configuring a managed client, then you also need to list the exact command lines that the master will be invoking on the managed client. You are guaranteed that the master will invoke one action at a time, so if you list two lines per action (full and non-full) you should be fine. Here's an example for the collect action:

```
/usr/bin/cback --full collect
/usr/bin/cback collect
```

Of course, you would have to list the actual path to the **cback** executable — exactly the one listed in the <cback_command> configuration option for your managed peer.

I hope that there is enough information here for interested users to implement something that makes them comfortable. I have resisted providing a complete example script, because I think everyone's setup will be different. However, feel free to write if you are working through this and you have questions.

Appendix E. Copyright

Copyright (c) 2005-2008
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

==