8

Backups and Removable Media

Data backups in Linux were traditionally done by running commands to archive and compress the files to back up, then writing that backup archive to tape. Choices for archive tools, compression techniques, and backup media have grown tremendously in recent years. Tape archiving has, for many, been replaced with techniques for backing up data over the network, to other hard disks, or to CDs, DVDs, or other low-cost removable media.

This chapter details some useful tools for backing up and restoring your critical data. The first part of the chapter details how to use basic tools such as tar, gzip, and rsync for backups.

IN THIS CHAPTER

Creating backup archives with tar

Compressing backups with gzip, bzip2, and lzop

Backing up over the network with SSH

Doing network backups with rsync

Making backup ISO images with mkisofs

Burning backup images to CD or DVD with cdrecord and growisofs

Backing Up Data to Compressed Archives

If you are coming from a Windows background, you may be used to tools such as WinZip and PKZIP, which both archive and compress groups of files in one application. Linux offers separate tools for gathering groups of files into a single archive (such as `tar`) and compressing that archive for efficient storage (`gzip`, `bzip2`, and `lzop`). However, you can also do the two steps together by using additional options to the `tar` command.

Creating Backup Archives with tar

The `tar` command, which stands for *tape archiver*, dates back to early Unix systems. Although magnetic tape was the common medium that `tar` wrote to originally, today `tar` is most often used to create an archive file that can be distributed to a variety of media.

The fact that the `tar` command is rich in features is reflected in the dozens of options available with `tar`. The basic operations of `tar`, however, are used to create a backup archive (`-c`), extract files from an archive (`-x`), compare differences between archives (`-d`), and update files in an archive (`-u`). You can also append files to (`-r` or `-A`) or delete files from (`-d`) an existing archive, or list the contents of an archive (`-t`).

> **NOTE** *Although the* `tar` *command is available on nearly all Unix and Linux systems, it behaves differently on many systems. For example, Solaris does not support* `-z` *to manage tar archives compressed in gzip format. The* `Star` *(ess-tar) command supports access control lists (ACLs) and file flags (for extended permissions used by Samba).*

As part of the process of creating a tar archive, you can add options that compress the resulting archive. For example, add `-j` to compress the archive in bzip2 format or `-z` to compress in gzip format. By convention, regular tar files end in `.tar`, while compressed tar files end in `.tar.bz2` (compressed with bzip2) or `.tar.gz` (compressed with gzip). If you compress a file manually with `lzop` (see `www.lzop.org`), the compressed tar file should end in `.tar.lzo`.

Besides being used for backups, tar files are popular ways to distribute source code and binaries from software projects. That's because you can expect every Linux and Unix-like system to contain the tools you need to work with tar files.

> **NOTE** *One quirk of working with the* `tar` *command comes from the fact that tar was created before there were standards regarding how options are entered. Although you can prefix* `tar` *options with a dash, it isn't always necessary. So you might see a command that begins* `tar xvf` *with no dashes to indicate the options.*

A classic example for using the `tar` command might combine old-style options and pipes for compressing the output; for example:

```
$ tar c *.txt | gzip -c > myfiles.tar.gz Make archive, zip it, and output
```

The example just shown illustrates a two-step process you might find in documentation for old Unix systems. The `tar` command creates (`c`) an archive from all `.txt` files in the current directory. The output is piped to the `gzip` command and output to stdout (`-c`), and then redirected to the `myfiles.tar.gz` file. Note that `tar` is one of the few commands which don't require that options be preceded by a dash (-).

New tar versions, on modern Linux systems, can **create the archive and compress the output** in one step:

```
$ tar czf myfiles.tar.gz *.txt   Create gzipped tar file of .txt files
$ tar czvf myfiles.tar.gz *.txt  Be more verbose creating archive
textfile1.txt
textfile2.txt
```

In the examples just shown, note that the new archive name (`myfiles.tar.gz`) must immediately follow the `f` option to `tar` (which indicates the name of the archive). Otherwise the output from `tar` will be directed to stdout (in other words, your screen). The `z` option says to do `gzip` compression, and `v` produces verbose descriptions of processing.

When you want to **return the files to a file system** (unzipping and untarring), you can also do that as either a one-step or two-step process, using the `tar` command and optionally the `gunzip` command:

```
$ gunzip -c myfiles.tar.gz | tar x            Unzips and untars archive
```

Or try the following command line instead:

```
$ gunzip myfiles.tar.gz ; tar xf myfiles.tar  Unzips then untars archive
```

To do that same procedure in one step, you could use the following command:

```
$ tar xzvf myfiles.tar.gz
textfile1.txt
textfile2.txt
```

The result of the previous commands is that the archived `.txt` files are copied from the archive to the current directory. The `x` option extracts the files, `z` uncompresses (unzips) the files, `v` makes the output, and `f` indicates that the next option is the name of the archive file (`myfiles.tar.gz`).

Using Compression Tools

Compression is an important aspect of working with backup files. It takes less disk space on your backup medium (CD, DVD, tape, and so on) or server to store compressed files. It also takes less time to transfer the archives to the media or download the files over a network.

While compression can save a lot of storage space and transfer times, it can significantly increase your CPU usage. You can consider using hardware compression on a tape drive (see `www.amanda.org/docs/faq.html#id346016`).

In the examples shown in the previous section, `tar` calls the `gzip` command. But `tar` can work with many compression tools. Out of the box on Fedora, `tar` will work with `gzip` and `bzip2`. A third compression utility we add to our toolbox is the `lzop` command, which can be used with `tar` in a different way. The order of these tools from fastest/least compression to slowest/most compression is: `lzop`, `gzip`, and `bzip2`.

If you are archiving and compressing large amounts of data, the time it takes to compress your backups can be significant. So you should be aware that, in general, `bzip2` may take about 10 times longer than `lzop` and only give you twice the compression. However, with each compression command, you can choose different compression levels, to balance the need for more compression with the time that compression takes.

To use the `tar` command with **bzip2 compression**, use the `-j` option:

```
$ tar cjvf myfiles.tar.bz2 *.txt   Create archive, compress with bzip2
```

You can also **uncompress (-j) a bzip2 compressed file** as you extract files (`-x`) using the `tar` command:

```
$ tar xjvf myfiles.tar.bz2   Extract files, uncompress bzip2 compression
```

The lzop compression utility is a bit less integrated into `tar`. Before you can use `lzop`, you might need to install the lzop package. To do **lzop compression**, you need the `--use-compress-program` option:

```
# yum install lzop
$ tar --use-compress-program=lzop -cf myfiles.tar.lzo *.txt
$ tar --use-compress-program=lzop -xf myfiles.tar.lzo
```

In the previous examples, the command line reverses the old syntax of `tar` with a switch before the command. For normal use and in other examples, we used the modern syntax of `tar` with no switch.

> **NOTE** *You may encounter .rar compressed files in the RAR format. This format seems to be popular in the world of peer-to-peer networks. RAR is a proprietary format so there is no widespread compressing tool. The FreshRPMs.net repository has a rar RPM package for some versions of Fedora. The* `unrar` *command, on the other hand, is more widely available. The Livna.org repository has an unrar RPM package for Fedora.*

Compressing with gzip

As noted, you can **use any of the compression commands alone** (as opposed to within the `tar` command line). Here are some examples of the `gzip` command to create and work with gzip-compressed files:

```
$ gzip myfile                gzips myfile and renames it myfile.gz
```

The following command provides the same result, with verbose output:

```
$ gzip -v  myfile            gzips myfile with verbose output
myfile: 86.0% -- replaced with myfile.gz
$ gzip -tv myfile.gz         Tests integrity of gzip file
myfile.gz:     OK
```

```
$ gzip -lv myfile.gz          Get detailed info about gzip file
method  crc   date  time    compressed    uncompressed  ratio uncompressed_name
defla 0f27d9e4 Jul 10 04:48     46785       334045       86.0%   myfile
```

Use any one of the following commands to **compress all files in a directory:**

```
$ gzip -rv mydir              Compress all files in a directory
mydir/file1: 39.1% -- replaced with mydir/file1.gz
mydir/file2: 39.5% -- replaced with mydir/file2.gz
$ gzip -1 myfile      Fastest compression time, least compression
$ gzip -9 myfile      Slowest compression time, most compression
```

Add a dash before a number from 1 to 9 to set the compression level. As illustrated above, `-1` is the fastest (least) and `-9` is the slowest (most) compression. The default for `gzip` is level 6. The `lzop` command has fewer levels: 1, 3 (default), 7, 8, and 9. Compression levels for `bzip2` behave differently.

To **uncompress a gzipped file**, you can use the `gunzip` command. Use either of the following examples:

```
$ gunzip -v myfile.gz         Unzips myfile.gz and renames it myfile
myfile.gz:        86.0% -- replaced with myfile
$ gzip -dv myfile.gz          Same as previous command line
```

Although the examples just shown refer to zipping regular files, the same options can be used to compress tar archives.

Compressing with bzip2

The **bzip2 command** is considered to provide the highest compression among the compression tools described in this chapter. Here are some examples of `bzip2`:

```
$ bzip2 myfile              Compresses file and renames it myfile.bz2
$ bzip2 -v myfile           Same as previous command, but more verbose
  myfile:  9.529:1, 0.840 bits/byte, 89.51% saved, 334045 in, 35056 out.
$ bunzip2 myfile.bz2        Uncompresses file and renames it myfile
$ bzip2 -d myfile.bz2       Same as previous command
$ bunzip2 -v myfile.bz2     Same as previous command, but more verbose
  myfile.bz2: done
```

Compressing with lzop

The `lzop` command behaves differently from `gzip` and `bzip2`. The `lzop` command is best in cases where compression speed is more important than the resulting compression ratio. When `lzop` compresses the contents of a file, it leaves the original file intact (unless you use `-U`), but creates a new file with a `.lzo` suffix. Use either of the following examples of the **lzop command to compress a file called myfile:**

```
$ lzop -v myfile              Leave myfile, create compressed myfile.lzo
compressing myfile into myfile.lzo
$ lzop -U myfile              Remove myfile, create compressed myfile.lzo
```

With **myfile.lzo** created, choose any of the following commands to **test, list, or uncompress the file**:

```
$ lzop -t myfile.lzo          Test the compressed file's integrity
$ lzop --info myfile.lzo      List internal header for each file
$ lzop -l myfile.lzo          List compression info for each file
method   compressed  uncompr. ratio uncompressed_name
LZO1X-1      59008     99468  59.3% myfile
$ lzop --ls myfile.lzo        Show contents of compressed file as ls -l
$ cat myfile | lzop > x.lzo   Compress standin and direct to stdout
$ lzop -dv myfile.lzo         Leave myfile.lzo, make uncompressed myfile
```

Unlike gzip and bzip2, lzop has no related command for unlzopping. Always just use the `-d` option to `lzop` to uncompress a file. If fed a list of file and directory names, the `lzop` command will compress all files and ignore directories. The original file name, permission modes, and timestamps are used on the compressed file as were used on the original file.

Listing, Joining, and Adding Files to tar Archives

So far, all we've done with `tar` is create and unpack archives. There are also options for listing the contents of archives, joining archives, adding files to an existing archive, and deleting files from an archive.

To **list an archive's contents**, use the `-t` option:

```
$ tar tvf myfiles.tar            List files from uncompressed archive
-rw-r--r-- root/root       9584 2007-07-05 11:20:33 textfile1.txt
-rw-r--r-- root/root       9584 2007-07-09 10:23:44 textfile2.txt
$ tar tzvf myfiles.tgz           List files from gzip compressed archive
```

If the archive were a tar archive compressed with lzop and named `myfile.tar.lzo`, you could **list that tar/lzop file's contents** as follows:

```
$ tar --use-compress-program=lzop -tf myfiles.tar.lzo  List lzo archives
```

To **concatenate one tar file to another**, use the `-A` option. The following command results in the contents of `archive2.tar` being added to the `archive1.tar` archive:

```
$ tar -Af archive1.tar archive2.tar
```

Use the `-r` option to **add one or more files to an existing archive**. In the following example, `myfile` is added to the `archive.tar` archive file:

```
$ tar rvf archive.tar myfile    Add a file to a tar archive
```

You can use wildcards to **match multiple files to add** to your archive:

```
$ tar rvf archive.tar *.txt     Add multiple files to a tar archive
```

Deleting Files from tar Archives

If you have a tar archive file on your hard disk, you can delete files from that archive. Note that you can't use this technique to delete files from tar output on magnetic tape. Here is an example of **deleting files from a tar archive:**

```
$ tar --delete file1.txt -f myfile.tar  Delete file1.txt from myfile.tar
```

Backing Up Over Networks

After you have backed up your files and gathered them into a tar archive, what do you do with that archive? The primary reason for having a backup is in case something happens (such as a hard disk crash) where you need to restore files from that backup. Methods you can employ to keep those backups safe include:

- **Copying backups to removable media** such as tape, CD, or DVD (as described later in this chapter)
- **Copying them to another machine over a network**

Fast and reliable networks, inexpensive high-capacity hard disks, and the security that comes with moving your data off-site have all made network backups a popular practice. For an individual backing up personal data or a small office, combining a few simple commands may be all you need to create efficient and secure backups. This approach represents a direct application of the Unix philosophy: joining together simple programs that do one thing to get a more complex job done.

Although just about any command that can copy files over a network can be used to move your backup data to a remote machine, some utilities are especially good for the job. Using OpenSSH tools such as ssh and scp, you can set up secure passwordless transfers of backup archives and encrypted transmissions of those archives.

Tools such as the `rsync` command can save resources by backing up only files (or parts of files) that have changed since the previous backup. With tools such as unison, you can back up files over a network from Windows, as well as Linux systems.

The following sections describe some of these techniques for backing up your data to other machines over a network.

> **NOTE** *A similar tool that might interest you is the* `rsnapshot` *command (*`yum install rsnapshot`*). The* `rsnapshot` *command (*`www.rsnapshot.org/`*) can work with* `rsync` *to make configurable hourly, daily, weekly, or monthly snapshots of a file system. It uses hard links to keep a snapshot of a file system, which it can then sync with changed files.*

Backing Up tar Archives Over ssh

OpenSSH (`www.openssh.org/`) provides tools to securely do remote login, remote execution, and remote file copy over network interfaces. By setting up two machines

to share encryption keys, you can transfer files between those machines without entering passwords for each transmission. That fact lets you create scripts to back up your data from an SSH client to an SSH server, without any manual intervention.

From a central Linux system, you can **gather backups from multiple client machines** using OpenSSH commands. The following example runs the `tar` command on a remote site (to archive and compress the files), pipes the tar stream to standard output, and uses the `ssh` command to catch the backup locally (over `ssh`) with `tar`:

```
$ mkdir mybackup ; cd mybackup
$ ssh francois@server1 'tar cf - myfile*' | tar xvf -
francois@server1's password: ******
myfile1
myfile2
```

In the example just shown, all files beginning with `myfile` are copied from the home directory of francois on server1 and placed in the current directory. Note that the left side of the pipe creates the archive and the right side expands the files from the archive to the current directory. (Keep in mind that ssh will overwrite local files if they exist, which is why we created an empty directory in the example.)

To reverse the process and **copy files from the local system to the remote system**, we run a local `tar` command first. This time, however, we add a `cd` command to put the files in the directory of our choice on the remote machine:

```
$ tar cf - myfile* | ssh francois@server1 \
         'cd /home/francois/myfolder; tar xvf -'
francois@server1's password: ******
myfile1
myfile2
```

In this next example, we're not going to untar the files on the receiving end, but instead **write the results to tgz files**:

```
$ ssh francois@server1 'tar czf - myfile*' | cat > myfiles.tgz
$ tar cvzf - myfile* | ssh francois@server1 'cat > myfiles.tgz'
```

The first example takes all files beginning with `myfile` from the francois user's home directory on server1, `tar`s and compresses those files, and directs those compressed files to the `myfiles.tgz` file on the local system. The second example does the reverse by taking all files beginning with `myfile` in the local directory and sending them to a `myfiles.tgz` file on the remote system.

The examples just shown are good for copying files over the network. Besides providing compression they also enable you to use any `tar` features you choose, such as incremental backup features.

Backing Up Files with rsync

A more feature-rich command for doing backups is `rsync`. What makes `rsync` so unique is the `rsync` algorithm, which compares the local and remote files one small block at a time using checksums, and only transfers the blocks that are different. This algorithm is so efficient that it has been reused in many backup products.

The `rsync` command can work either on top of a remote shell (ssh), or by running an `rsyncd` daemon on the server end. The following example uses `rsync` over ssh to **mirror a directory**:

```
$ rsync -avz --delete chris@server1:/home/chris/pics/ chrispics/
```

The command just shown is intended to mirror the remote directory structure (`/home/chris/pics/`) on the local system. The `-a` says to run in archive mode (recursively copying all files from the remote directory), the `-z` option compresses the files, and `-v` makes the output verbose. The `--delete` tells `rsync` to delete any files on the local system that no longer exist on the remote system.

For ongoing backups, you can have `rsync` do seven-day incremental backups. Here's an example:

```
# mkdir /var/backups
# rsync --delete --backup                        \
    --backup-dir=/var/backups/backup-`date +%A` \
    -avz chris@server1:/home/chris/Personal/    \
    /var/backups/current-backup/
```

When the command just shown runs, all the files from `/home/chris/Personal` on the remote system server1 are copied to the local directory `/var/backups/current-backup`. All files modified today are copied to a directory named after today's day of the week, such as `/var/backups/backup-Monday`. Over a week, seven directories will be created that reflect changes over each of the past seven days.

Another trick for rotated backups is to **use hard links instead of multiple copies** of the files. This two-step process consists of rotating the files, then running `rsync`:

```
# rm -rf /var/backups/backup-old/
# mv /var/backups/backup-current/ /var/backups/backup-old/
# rsync --delete --link-dest=/var/backups/backup-old -avz \
   chris@server1:/home/chris/Personal/ /var/backups/backup-current/
```

In the previous procedure, the existing `backup-current` directory replaces the `backup-old` directory, deleting the two-week-old full backup with last-week's full backup. When the new full backup is run with `rsync` using the `--link-dest` option,

if any of the files being backed up from the remote `Personal` directory on server1 existed during the previous backup (now in `backup-old`), a hard link is created between the file in the `backup-current` directory and `backup-old` directory.

You can save a lot of space by having hard links between files in your `backup-old` and `backup-current` directory. For example, if you had a file named `file1.txt` in both directories, you could check that both were the same physical file by listing the files' inodes as follows:

```
$ ls -i /var/backups/backup*/file1.txt
260761  /var/backups/backup-current/file1.txt
260761  /var/backups/backup-old/file1.txt
```

Backing Up with unison

Although the `rsync` command is good to back up one machine to another, it assumes that the machine being backed up is the only one where the data is being modified. What if you have two machines that both modify the same file and you want to sync those files? Unison is a tool that will let you do that.

It's common for people to want to work with the same documents on their laptop and desktop systems. Those machines might even run different operating systems. Because unison is a cross-platform application, it can let you **sync files** that are on both Linux and Windows systems. To use unison in Fedora, you must install the unison package (type the `yum install unison` command).

With unison, you can define two *roots* representing the two paths to synchronize. Those roots can be local or remote over ssh. For example:

```
$ unison /home/francois ssh://francois@server1//home/fcaen
$ unison /home/francois /mnt/backups/francois-homedir
```

Unison contains both graphical and command-line tools for doing unison backups. It will try to run the graphical version by default. This may fail if you don't have a desktop running or if you're launching unison from within `screen`. To **force unison to run in command line mode**, add the `-ui text` option as follows:

```
$ unison /home/francois ssh://francois@server1//home/fcaen -ui text
Contacting server...
francois@server1's password:
Looking for changes
   Waiting for changes from server
Reconciling changes
local         server1
newfile ---->           memo.txt  [f] y
Propagating updates
   ...
```

The unison utility will then compare the two roots and for each change that occurred since last time, ask you what you want to do. In the example above, there's a new file called `memo.txt` on the local system. You are asked if you want to proceed with the update (in this case, copy `memo.txt` from the local machine to server1). Type `y` to do the updates.

If you trust unison, add `-auto` to make it **take default actions without prompting you**:

```
$ unison /home/francois ssh://francois@server1//home/fcaen -auto
```

There is no man page for unison. However, you can view `unison` options using the `-help` option. You can also display and page through the `unison` manual using the `-doc all` option as shown here:

```
$ unison -help                  See unison options
$ unison -doc all | less        Display unison manual
```

If you find yourself synchronizing two roots frequently, you can **create a profile**, which is a series of presets. In graphical mode, the default screen makes you create profiles. Profiles are stored in `.prf` text files in the `~/.unison/` directory. They can be as simple as the following:

```
root = /home/francois
root = ssh://francois@server1//home/fcaen
```

If this is stored in a profile called `fc-home.prf`, you can invoke it simply with the following command line:

```
$ unison fc-home
```

Backing Up to Removable Media

The capacity of CDs and DVDs, and the low costs of those media, has made them attractive options as computer backup media. Using tools that commonly come with Linux systems, you can gather files to back up into CD or DVD images and burn those images to the appropriate media.

Command line tools such as `mkisofs` (for creating CD images) and `cdrecord` (for burning images to CD or DVD) once provided the most popular interfaces for making backups to CD or DVD. Now there are many graphical front-ends to those tools you could also consider using. For example, GUI tools for mastering and burning CDs/DVDs include K3b (the KDE CD and DVD Kreator) and Nautilus (GNOME's file manager that offers a CD-burning feature). Other GUI tools for burning CDs include gcombust, X-CD-Roast, and graveman.

The commands for creating file system images to back up to CD or DVD, as well as to burn those images, are described in this section.

Creating Backup Images with mkisofs

Most data CDs and DVDs can be accessed on both Windows and Linux systems because they are created using the ISO9660 standard for formatting the information on those discs. Because most modern operating systems need to save more information about files and directories than the basic ISO9660 standard includes, extensions to that standard were added to contain that information.

Using the `mkisofs` command, you can back up the file and directory structure from any point in your Linux file system and produce an ISO9660 image. That image can include the following kinds of extensions:

- **System Use Sharing Protocol** (SUSP) are records identified in the Rock Ridge Interchange Protocol. SUSP records can include Unix-style attributes, such as ownership, long file names, and special files (such as character devices and symbolic links).
- **Joliet** directory records store longer file names in a form that makes them usable to Windows systems.
- **Hierarchical File System** (HFS) extensions allow the ISO image to appear as an HFS file system, which is the native file system for Macintosh computers. Likewise, Data and Resource forks can be added in different ways to be read by Macs.

When you set out to create your ISO image, consider where you will ultimately need to access the files you back up using `mkisofs` (Linux, Windows, or Macs). Once the image is created, it can be used in different ways, the most obvious of which is to burn the image to a CD or DVD.

Besides being useful in producing all or portions of a Linux file system to use on a portable medium, `mkisofs` is also useful for creating live CDs/DVDs. It does this by adding boot information to the image that can launch a Linux kernel or other operating system, bypassing the computer's hard drive.

> **NOTE** *Although you can still use the* `mkisofs` *command in Fedora,* `mkisofs` *is now a pointer to* `genisoimage`*. The* `genisoimage` *command was derived from* `mkisofs`*, which was part of the* `cdrtools` *package (see* `http://cdrecord.berlios.de`*). Development of* `genisoimage` *is part of the cdrkit project (*`www.cdrkit.org`*).*

Because most Linux users store their personal files in their home directories, a common way to use `mkisofs` to back up files is to back up everything under the `/home` directory. Here are some examples of using `mkisofs` to **create an ISO image from all files and directories under the /home directory**:

```
# cd /tmp
# mkisofs -o home.iso /home              Create basic ISO9660 image
# mkisofs -o home2.iso -J -R /home       Add Joliet Rock Ridge extensions
# mkisofs -o home3.iso -J -R -hfs /home Also add HFS extensions
```

In each of the three examples above, all files and directories beneath the `/home` directory are added to the ISO image (`home.iso`). The first example has no extensions, so all file names are converted to DOS-style naming (8.3 characters). The second example uses Joliet and Rock Ridge extensions, so file names and permissions should appear as they did on the original Linux system when you open the ISO on a Linux or Windows system. The last example also makes the files on the image readable from a Mac file system.

You can **have multiple sources added to the image**. Here are some examples:

```
# mkisofs -o home.iso -R -J music/ docs/ \  Multiple directories/files
          chris.pdf /var/spool/mail
# mkisofs -o home.iso -J -R              \  Graft files on to the image
     -graft-points Pictures/=/var/pics/  \
     /home/chris
```

The first example above shows various files and directories being combined and placed on the root of the ISO image. The second example grafts the contents of the `/var/pics` directory into the `/home/chris/Pictures` directory. As a result, on the CD image the `/Pictures` directory will contain all content from the `/var/pics` directory.

Adding information into the header of the ISO image can help you identify the contents of that image later. This is especially useful if the image is being saved or distributed online, without a physical disc you can write on. Here are some examples:

```
# mkisofs -o /tmp/home.iso -R -J            \  Add header info to ISO
     -p www.handsonhistory.com              \
     -publisher "Swan Bay Folk Art Center"  \
     -V "WebBackup"                         \
     -A "mkisofs"                           \
     -volset "1 of 4 backups, July 30, 2007" \
     /home/chris
```

In the example above, `-p` indicates the preparer ID, which could include a phone number, mailing address, or web site for contacting the preparer of the ISO image. With the option `-publisher`, you can indicate a 128-character description of the preparer (possibly the company or organization name). The `-V` indicates the volume ID. Volume ID is important because in many Linux systems, this volume ID is used to mount the CD when it is inserted. For example, in the command line shown above, the CD would be mounted on `/media/WebBackup` in Fedora and other Linux systems. The `-A` option can be used to indicate the application used to create the ISO image. The `-volset` option can contain a string of information about a set of ISO images.

When you have created your ISO image, and before you burn it to disc, you **can check the image** and make sure you can access the files it contains. Here are ways to check it out:

```
# volname home.iso              Display volume name
WebBackup
# isoinfo -d -i home.iso        Display header information
CD-ROM is in ISO 9660 format
System id: LINUX
```

```
Volume id: WebBackup
Volume set id: All Website material on November 2, 2007
Publisher id: Swan Bay Folk Art Center
Data preparer id: www.handsonhistory.com
Application id: mkisofs
Copyright File id:
Abstract File id:
Bibliographic File id:
Volume set size is: 1
Volume set sequence number is: 1
Logical block size is: 2048
Volume size is: 23805
Joliet with UCS level 3 found
Rock Ridge signatures version 1 found
```

You can see a lot of the information entered on the `mkisofs` command line when the image was created. If this had been an image that was going to be published, we might also have indicated the locations on the CD of a copyright file (`-copyright`), abstract file (`-abstract`), and bibliographic file (`-biblio`). Provided that the header is okay, you can next try **accessing files on the ISO image by mounting it:**

```
# mkdir /mnt/myimage                      Create a mount point
# mount -o loop home.iso /mnt/myimage     Mount the ISO in loopback
# ls -l /mnt/myimage                      Check the ISO contents
# umount /mnt/myimage                     Unmount the image when done
```

Besides checking that you can access the files and directories on the ISO, make sure that the date/time stamps, ownership, and permissions are set as you would like. That information might be useful if you need to restore the information at a later date.

Burning Backup Images with cdrecord

The `cdrecord` command is the most popular Linux command line tool for burning CD and DVD images. After you have created an ISO image (as described earlier) or obtained one otherwise (such as downloading an install CD or live CD from the Internet), `cdrecord` makes it easy to put that image on a disc.

> **NOTE** *As of Fedora 7,* `cdrecord` *was replaced with the* `wodim` *command. The* `wodim` *command was created from the* `cdrecord` *code base and still supports most of the same options. If you run* `cdrecord`*, you will actually be running* `wodim` *in this Fedora release. If you have problems with that utility, contact the CDRkit project (*`http://cdrkit.org`*).*

There is no difference in making a CD or DVD ISO image, aside from the fact that a DVD image can obviously be bigger than a CD image. Check the media you have for their capacities. A CD can typically hold 650MB, 700MB, or 800MB, whereas mini CDs can hold 50MB, 180MB, 185MB, or 193MB. Single-layer DVDs hold 4.7GB, while double-layer DVDs can hold 8.4GB.

NOTE *Keep in mind, however, that CD/DVD manufacturers list their capacities based on 1000 KB per 1 MB, instead of 1024 KB. Type* `du --si home.iso` *to list the size of your ISO, instead of* `du -sh` *as you would normally, to check if your ISO will fit on the media you have.*

Before you begin burning your image to CD or DVD, **check that your drive supports CD/DVD burning** and determine the address of the drive. Use the `--scanbus` option to `cdrecord` to do that:

```
# cdrecord --scanbus          Shows a drive that cannot do burning
scsibus0:
       0,0,0  0) 'SAMSUNG ' 'DVD-ROM SD-616E ' 'F503' Removable CD-ROM
       0,0,0  1) *
       0,0,0  2) *
              ...
# cdrecord --scanbus          Shows a drive that can burn CDs or DVDs
scsibus0:
       0,0,0  0) 'LITE-ON ' 'DVDRW SOHW-1633S' 'BS0C' Removable CD-ROM
       0,0,0  1) *
       0,0,0  2) *
              ...
```

In the two examples shown, the first indicates a CD/DVD drive that only supports reading and cannot burn CDs (DVD-ROM and CD-ROM). The second example shows a drive that can burn CDs or DVDs (DVDRW). Insert the medium you want to record on. Assuming your drive can burn the media you have, here are some simple `cdrecord` commands for **burning a CD or DVD images**:

```
# cdrecord -dummy home.iso      Test burn without actually burning
# cdrecord -v home.iso          Burn CD (default settings) in verbose
# cdrecord -v speed=24 home.iso Set specific speed
# cdrecord -pad home.iso        Can't read track so add 15 zeroed sectors
# cdrecord -eject home.iso      Eject CD/DVD when burn is done
# cdrecord /dev/cdrw home.iso   Identify drive by device name (may differ)
# cdrecord dev=0,2,0 home.iso   Identify drive by SCSI name
```

The `cdrecord` command can also **burn multi-session CDs/DVDs**. Here is an example:

```
# cdrecord -multi home.iso     Start a multi-burn session
# cdrecord -msinfo             Check the session offset for next burn
Using /dev/cdrom of unknown capabilities
0,93041
# mkisofs -J -R -o new.iso \   Create a second ISO to burn
   -C 0,93041 /home/chris/more Indicate start point and new data for ISO
# cdrecord new.iso             Burn new data to existing CD
```

You can use multiple `-multi` burns until the CD is filled up. For the final burn, don't use `-multi`, so that the CD will be closed.

Making and Burning DVDs with growisofs

Using the `growisofs` command, you can **combine the two steps of gathering files into an ISO image (mkisofs) and burning that image to DVD (cdrecord)**. Besides saving a step, the `growisofs` command also offers the advantage of keeping a session open by default until you close it, so you don't need to do anything special for multi-burn sessions.

Here is an example of some `growisofs` commands for a **multi-burn session**:

```
# growisofs -Z /dev/dvd -R -J /home/chris      Master and burn to DVD
# growisofs -Z /dev/dvd -R -J /home/francois   Add to burn
# growisofs -M /dev/dvd=/dev/zero              Close burn
```

If you want to add options when creating the ISO image, you can simply add `mkisofs` options to the command line. (For example, see how the `-R` and `-J` options are added in the above examples.)

If you want to **burn a DVD image using growisofs**, you can use the **-dvd-compat** option. Here's an example:

```
# growisofs -dvd-compat -Z /dev/dvd=image.iso Burn an ISO image to DVD
```

The `-dvd-compat` option can improve compatibility with different DVD drives over some multi-session DVD burning procedures.

Summary

Linux and its predecessor Unix systems handled data backups by combining commands that each handled a discrete set of features. Backups of your critical data can still be done in this way. In fact, many of the tools you can use will perform more securely and efficiently than ever before.

The tape archiver utility (`tar` command) has expanded well beyond its original job of making magnetic tape backups of data files. Because nearly every Linux and UNIX system includes `tar`, it has become a standard utility for packaging software and backing up data to compressed archives. Those archives can then be transported and stored in a variety of ways.

To move backed up data to other machines over a network, you can use remote execution features of OpenSSH tools (such as `ssh`). You can also use an excellent utility called `rsync`. With `rsync`, you can save resources by only backing up files (or parts of files) that have changed.

Inexpensive CDs and DVDs have made those media popular for doing personal and small-office backups. The `mkisofs` command can create file systems of backed up data in ISO9660 format that can be restored on a variety of systems (Linux, Windows, or Mac). Once `mkisofs` command has created an ISO image, the image can be burned to CD or DVD using the `cdrecord` or `growisofs` command.