SYBEX Sample Chapter

The Mac® OS X Command Line: Unix Under the Hood

Kirk McElhearn

Chapter 5: Working with Files and Directories

Copyright © 2004 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

ISBN: 0-7821-4354-7

SYBEX and the SYBEX logo are either registered trademarks or trademarks of SYBEX Inc. in the USA and other countries.

TRADEMARKS: Sybex has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer. Copyrights and trademarks of all products and services listed or described herein are property of their respective owners and companies. All rules and laws pertaining to said copyrights and trademarks are inferred.

This document may contain images, text, trademarks, logos, and/or other material owned by third parties. All rights reserved. Such material may not be copied, distributed, transmitted, or stored without the express, prior, written consent of the owner.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturers. The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Sybex Inc.
1151 Marina Village Parkway
Alameda, CA 94501
U.S.A.
Phone: 510-523-8233
www.sybex.com

Chapter 5

Working with Files and Directories

One of the strongest features of the Mac OS X Finder, and of other graphical user interface operating systems, is its ability to make file management actions so intuitive that they seem effortless. When you drag a file from one window to another, for example, the Finder is merely putting a graphical face on a basic action, that of issuing a command to move a file from one location to another. Figure 5.1 is an example of what you see in the Finder.

But what you would type to carry out this same operation from the command line is this:

```
$ mv ~/My\ Folder/My\ File.txt ~/My\ Other\ Folder
```

FIGURE 5.1
Moving a file in the Finder

You have to admit, there's something to be said for dragging and dropping files in Finder windows!

But the command line offers many advantages over using the Finder for moving, copying, or deleting files and folders. Here are just a few:

- You can copy or move files from one directory to a distant directory, without having to open any windows.
- You can easily copy or move multiple files using wildcards. You can even select which files to copy according to certain attributes, such as parts of filenames or extensions.
- You can copy or move files that are hidden by the Finder.
- You can make files invisible with one simple command.
- You can rename files and directories in a jiffy. You can even move and rename a file or directory with just one command.
- You can delete files or directories quickly (but irreversibly—see the section "Removing Files with *rm*" below for a warning on deleting files).
- You can delete files that the Finder refuses to delete. Occasionally, a recalcitrant file that you have placed in the Trash just won't go away. Using the command line, you can eliminate it for good.

As you get more familiar with working via the command line, you will discover more advantages to using Terminal.

A Caveat on Moving and Copying Files

Mac OS has historically used a unique way of saving files. Many files are in two parts, called the *data fork* and the *resource fork*. Back in the days of Mac OS 9 and earlier, the data fork contained data (the contents of a file, or code for applications) and the resource fork contained settings, icons, and other information. Most files used this two-part system; if you have ever copied files to a PC disk and looked on a Windows computer, you have seen additional folders copied together with your files.

When copying or moving files containing a resource fork, this can be a problem: some files still contain information in their resource fork, especially those files used by Mac OS 9 or Classic applications.

If you are using files with Mac OS 9 or Classic, the `cp` and `mv` commands may strip any resources the files contain. (The `mv` command is usually safe, and only strips resource forks when moving a file to another volume; this is, in essence, copying the file, since it creates a new copy of the file on the other volume and then deletes the original.) Some applications need these resource forks, and stripping them may cause files you've moved or copied with these commands to be unusable. But Apple created some additional commands, called `CpMac` and `MvMac` (the capitalization is important here), to resolve these issues. `CpMac` and `MvMac` are installed with the Developer Tools. (For more on using the Developer Tools, see Interlude 8, "Using the Developer Tools.") These command-line tools allow you to retain resource forks when copying and moving files and applications, ensuring that everything you copy or move with them remains usable.

Copying Files with *cp*

The `cp` (copy) command does exactly what its name suggests: it copies files from one location to another. At its simplest implementation, `cp` copies a file (specified by its filename, with either a relative or absolute path) to a directory (also specified by its name, with either a relative or absolute path). The basic form is as follows:

```
cp source destination
```

With this in mind, let's look at a few examples of copying files.

```
$ cp ~/Documents/MyFile.rtf ~/Public
```

In the above example, I copied a file from the `Documents` directory in my home directory to the `Public` directory, a location where any user can access files. As you can see in the command, the first part, `cp`, is the command, the second, `~/Documents/MyFile.rtf`, is the source, and the third, `~/Public`, is the destination. Both the source and destination in this example use absolute filepaths; I could be anywhere in the file system when running this command.

However, if I were already in the `Documents` directory, I wouldn't need to use an absolute filepath for the source. Since it's perfectly legal to mix absolute and relative filepaths in a command, I could merely type the following:

```
$ cp MyFile.rtf ~/Public
```

Let's say I'm in my `Documents` directory and want to copy a file there from my `Public` directory. I could use the following command, using the `.` and `..` shortcuts:

```
$ cp ../Public/MyFile.rtf .
```

The source, `../Public/MyFile.rtf`, tells the shell to look at the `Public` directory, which is a subfolder of the next directory up in the hierarchy (`..`, or my home directory), and to copy the file called `MyFile.rtf`. The destination, `.`, is the shortcut for the current directory.

When copying files in this simple form, the source is a filename and the destination a directory. (The destination can also be a filename; see below for more ways to use `cp`.) But the source can also be multiple filenames. When executing this command, the shell checks the contents of the directory you refer to, making sure that the file or files exist. If there are several files listed and they all exist, then the shell goes ahead and copies them all to the destination. You can run a command like this:

```
$ cp MyFile1.rtf MyFile2.rtf MyFile3.rtf ~/Public
```

and copy the three files after the `cp` command to the destination directory.

Using Wildcards to Copy Files

For an even shorter version of this command, you can use a wildcard and save your typing fingers:

```
$ cp MyFile* ~/Public
```

Warning: Copying Files Replaces Existing Files

By default, the cp command replaces any like-named files in the destination—unlike the Finder, which gives you an alert, asking if you really want to replace them. The same goes for the mv command (see the section "Moving Files and Directories with *mv*" below). This is one of the dangers of using the command line. New actions call for new habits, and the safest way to work with these two commands is to use the -i (interactive) option, which tells the shell to ask you if any files with the same name are present. When using this option, type y for yes (to replace files) and n for no.

Here's an example:

```
$ cp -i MyFile1.rtf New_Directory/
overwrite New_Directory/MyFile1.rtf? y
$
```

I typed y when Terminal asked me if I wanted to overwrite the file. If I typed n, or pressed Return, at the overwrite question, the command would stop.

The * wildcard tells the shell to look for all files whose names begin with MyFile and copy them to ~/Public. Of course, if you have 10 files like that, all 10 will be copied. If you only want the first three copied, you need to enter each name individually, or you could use the following command:

```
$ cp MyFile[1-3].rtf ~/Public
```

If you want to copy all .rtf files from the source directory, you can use the following:

```
$ cp *.rtf ~/Public
```

This will copy all files ending with .rtf to the Public folder. You can use the asterisk wildcard at any location in a filename. Table 5.1 demonstrates some ways it can be used.

For a more thorough presentation of wildcards, see Interlude 6, "Wildcards and Globbing."

Table 5.1: Examples of Using the Asterisk As a Wildcard

Wildcard Usage	What the Shell Matches
MyFile*.rtf	This string tells the shell to look for any file beginning with MyFile and ending with .rtf. It will copy, for example, MyFile1.rtf, as well as MyFileBackup.rtf.
My*.rtf	This string tells the shell to look for any file beginning with My and ending with .rtf. It will also copy MyReport.rtf or MyBicycle.rtf.
*File.rtf	This string tells the shell to look for any file ending with File.rtf. It will copy OldFile.rtf, NewFile.rtf, etc.
My*le[1-3].rtf	This string tells the shell to look for any file beginning with My, followed by any characters, then by le1, le2, or le3, then .rtf. It will copy MyVeryOldFile1.rtf, but not MyFileBackup1.rtf.

Hands On: Copying All the Files in a Directory

As seen in Table 5.1, wildcards save you from typing lots of filenames. You can go the ultimate distance with the * wildcard and use it to copy all the files from a directory to another directory. To copy all the files from the current working directory (the directory Terminal is currently in), run the following:

```
$ cp * [destination]
```

For example, to copy all the files in your `Pictures` directory to your home directory, you can run this while in the `Pictures` directory:

```
$ cp * ~
```

If you are not in the directory you want to copy from, you merely need to specify that directory in the source, like this:

```
$ cp ~/Pictures/* ~
```

Not only does this save you a lot of keystrokes, but you'll find that it can be even quicker than using the Finder.

Copying a File and Changing Its Name

In the above examples, the sources used were files and the destinations directories. But the destination can also be a filename. This is useful if you want to copy a file and change its name at the same time. For example, to copy `MyFile.rtf` to your `Public` folder, renaming it `MyFile1.rtf`, run the following command:

```
$ cp MyFile.rtf ~/Public/MyFile1.rtf
```

Unless you want to change the name, you won't need to specify a filename in the destination; a directory will do.

You can do the same thing to make a copy of a file, with a different name, in the same folder. Just run the command like this:

```
$ cp MyFile.rtf MyFile1.rtf
```

The system will make a copy of the file, with the new name.

Preserving File Information while Copying

When you copy a file with `cp`, some of the information about the file may be lost or changed. This notably includes the file's modification date and time. The `cp` command normally updates this information when copying, but if you run this command with the `-p` (preserve) option, this will be preserved. This is very useful when you are concerned about knowing which of your files is the newest or most recently modified.

In the following example, I had a file called `testfile`, which I copied normally as `testfile1`, then again using `cp -p` as `testfile2`. You can see that in the first case the modification time was changed and in the second it was preserved.

```
-rw-r--r--   1 kirk   staff        0 Jan 13 09:49 testfile
-rw-r--r--   1 kirk   staff        0 Jan 13 10:19 testfile1
-rw-r--r--   1 kirk   staff        0 Jan 13 09:49 testfile2
```

This option preserves more than just the modification time. According to the `cp` man page, it "causes `cp` to preserve in the copy as many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions." Not all this information is preserved in all cases. For more on using this option, see the `cp` man page.

Copying Directories with *cp*

You can use `cp` to copy directories as well as files, but it works a bit differently. For `cp` to work with directories, it needs the `-R` (recursive) option. This tells the command to copy not only the directory specified, but all subdirectories it contains as well as any other contents. To copy a directory, you need to run the following:

```
$ cp -R [source] [destination]
```

All the other options and ways of copying, shown above for files, work the same with directories. Note, however, that while you rename the directory while copying—giving the new directory a different name than the original—you cannot change the name of its subdirectories or other contents.

ON THE COMMAND LINE

Overview of the *cp* Command

COMMAND SYNTAX

```
cp [option(s)] source destination
```

FINDER EQUIVALENT

Copy and Paste.

OVERVIEW OF OPTIONS FOR *CP*

- `cp` This command copies the source argument to the destination. You can use multiple sources or wildcards. You can specify a directory as the destination, or specify a path containing a file name, if you wish to change the name of the file being copied.
- `-p` This option causes `cp` to preserve in the copy as many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions.
- `-R` If the source is a directory, with this option tells `cp` to copy the directory and all its subdirectories.
- `-i` This is the interactive option. The shell will ask you to confirm before copying a file that would overwrite an existing file. If you type `y`, the copy will continue. If you type `n`, it will stop.

GETTING MORE INFORMATION

To display the man page and learn more about `cp`, and the options available, type `man cp` in Terminal. Several options are available that can be useful in certain situations.

The Incredible Shrinking Files

The cp command does not copy the resource forks of files. For most files, this doesn't make a difference, but some programs still put information in resources. One example is graphics programs that may put thumbnail images in the resources of picture files.

You can use cp to remove these resources, making the files much smaller, so you can send them more easily by e-mail or just save space. (If you have a lot of pictures, you can save several megabytes by removing these thumbnails.)

To do this, just copy a file, giving it a new name.

```
$ cp Perceval.jpg PercevalCopy.jpg
```

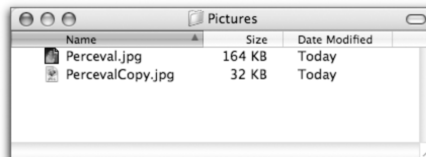
In the above example, I copied a picture of my son Perceval, renaming it, in the same directory. If you look in the Finder, you can see the difference, as shown here:

The original file displays a thumbnail and the copy just shows a standard JPEG icon.

Curiously, when examining the file info in Terminal, there seems to be no difference:

```
$ ls -l
-rw-r--r--  1 kirk  staff  30819 Nov 14 18:06 Perceval.jpg
-rw-r--r--  1 kirk  staff  30819 Nov 14 18:07 PercevalCopy.jpg
```

But as you can see here, the Finder shows that the difference is significant:



This is because Terminal does not see the resource fork of the files, and calculates the size only by looking at their data. However, running ls -ls shows the difference (the -ls option adds a column at the beginning of the line with the number of 512-byte blocks each file takes up):

```
$ ls -ls
```

Continued on next page

The Incredible Shrinking Files *(continued)*

```
328 -rw-r--r--  1 kirk  staff  30819 Nov 14 18:06 Perceval.jpg
 64 -rw-r--r--  1 kirk  staff  30819 Nov 14 18:07 PercevalCopy.jpg
```

Removing this thumbnail reduces the size of the file fivefold. If you are sending a number of pictures by e-mail, this can save a lot of time. However, if you want to strip resource forks from many files at once, you can use the `cp` command to copy the files to another folder. Run the command like this:

```
$ cp *.jpg /folder
```

sending the copies to another folder. In this manner using the * wildcard to match all filenames with a `jpg` extension the resource forks of all the files are removed.

Moving Files and Directories with *mv*

The `mv` (**move**) command moves files or directories from one location to another. When these locations are on the same volume, `mv` works the same as when you drag files from one Finder window to another. If you run the `mv` command across volumes, the files or directories are removed from their original location, whereas the Finder merely copies them. The `mv` command acts like a cut-and-paste operation, cutting the file or directory from its original location, and pasting it in its new location. This command works almost exactly like the `cp` command, the main exception being that you never have to use the `-R` option to move directories. To run this command, do the same things as described in the previous section, substituting `mv` for `cp`.

```
$ mv MyFile.rtf ~/Public/MyFile.rtf
```

In this example, the file called `MyFile.rtf` is moved to my `Public` folder. The operating system first writes the file then, after checking to make sure the copied file was written correctly, deletes the original.

Hands On: Renaming Files in a Jiffy

As we saw earlier, the `mv` command lets you choose a new name for a file at the destination. But you can also quickly rename a file with this command. All you need to do is move a file to the same directory it's in while changing its name. Run the following:

```
$ mv OldFileName NewFileName
```

The `mv` command takes `OldFileName` and replaces it with `NewFileName` in the same directory (since no different path is specified in the destination). But it gets even better: this command does not differentiate between files and directories, and you can use `mv` to rename a directory, retaining all its contents, no matter how many subdirectories or files it contains.

```
$ mv Photos Photos-copy
```

But be warned: as with many Unix commands, there is little room for error. You are safest copying files and directories if you want to change their names. When doing this, you are left with both the original and the copy, whereas `mv` leaves you with only the copy.

I must warn you that by default, the `mv` command (like `cp` discussed earlier) replaces any like-named files in the destination, unlike the Finder, which gives you an alert asking if you really want to replace them. You can run the `mv` command with the `-i` (interactive) option, as with many commands, to protect against this. If you run the command in the following manner, you will get an alert if a file exists with the same name:

```
$ mv -i MyFile.rtf ~/Public
overwrite /Users/kirk/Public/MyFile.rtf? y
```

Type `y` to overwrite the file, or type `n` or Return to stop the move operation and keep the file in the destination.

ON THE COMMAND LINE

Overview of the *mv* Command

COMMAND SYNTAX

```
mv [option(s)] source destination
```

FINDER EQUIVALENT

Cut and Paste; changing filenames by clicking and editing them.

OVERVIEW OF OPTIONS FOR *MV*

`mv` This command moves the source argument to the destination. You can use multiple sources or wildcards. You can specify a directory as the destination, or specify a path containing a filename, if you wish to change the name of the file being moved.

`-i` This is the interactive option. The shell will ask you to confirm before moving a file that would overwrite an existing file. If you type `y`, the move operation will continue. If you type `n`, or press Return, it will stop.

GETTING MORE INFORMATION

To display the man page and learn more about `mv`, and the options available, type `man mv` in Terminal.

Creating Directories with *mkdir*

We have seen above how to move and copy files from one directory to another, but you will also need to create directories to put these files in. The `mkdir` (**m**a**k**e **dir**ectory) command is very easy to use. Here's an example:

```
$ mkdir Test
```

This command creates a new directory, called `Test`, in the current directory. Since the filename does not begin with a `/`, the shell knows that you are specifying a relative path. If you want to create

the same directory in, say, your `Documents` directory, you could run the above command after using `cd` to move to that directory, or run the following from anywhere:

```
$ mkdir ~/Documents/Test
```

The `mkdir` command can also make several directories at a time. If you want to create three directories, called `Test1`, `Test2`, and `Test3`, you can run the following:

```
$ mkdir Test1 Test2 Test3
```

This command will create the three directories in the current working directory. To do the same using absolute paths, the command looks like this:

```
$ mkdir ~/Documents/Test1 ~/Documents/Test2 ~/Documents/Test3
```

If you want to create directories in hierarchy, `mkdir` can help as well. The only condition is that you set up your command in hierarchical order, creating the parent directory before the subdirectory, and before the sub-subdirectory, etc. If you want to create a series of directories and subdirectories like this:

```
Test1/Test2/Test3
```

you need to run the command as follows:

```
$ mkdir -p Test1/Test2/Test3
```

On The Command Line

Overview of the *mkdir* Command

Command Syntax

```
mkdir directory ...
```

Finder Equivalent

New Folder.

Overview of Options for *mkdir*

`mkdir` This command creates a new directory. You can specify either a relative or an absolute path for the directories you create.

`-p` This option tells `mkdir` to create all intermediate directories when you specify a hierarchy of directories.

Getting More Information

To display the man page and learn more about `mkdir`, and the options available, type `man mkdir` in Terminal. There are several options involving permissions for the directories created, which may be useful in some cases.

The -p (path) option tells the command to create each intermediate directory as required. The command first creates the Test1 directory, then the Test2 subdirectory, and finally, further down in the hierarchy, Test3.

Removing Directories with *rmdir*

The rmdir (**re**move **dir**ectories) command is self-explanatory: it lets you remove directories, deleting them forever. Like the rm command (covered in the next section), which works on files, rmdir is very powerful: once you remove a directory in this manner, there is no getting it back. (See the section "A Safer Way to Remove Files" below for a way to safeguard this command.)

However, rmdir only works with empty directories. You may find it easier to use rm for both files and directories in all cases—after all, it's much easier to use just one command instead of two, and if you apply the safeguard shown below, you need only set it for one command: rm.

To delete an empty directory, run the following:

```
$ rmdir Files
```

If you try to remove a directory that contains files or other directories, Terminal will display the following:

```
$ rmdir Files
rmdir: Files: Directory not empty
```

On The Command Line

Overview of the *rmdir* Command

Command Syntax

```
rmdir [option] [directory1] [directory2] ...
```

Finder Equivalent

Move (Folder) to Trash (though rmdir deletes the directory immediately).

Overview of Options for *rmdir*

rmdir	This command removes directories. You can specify a single directory or several directories as arguments.
-p	This option tells rmdir to remove all intermediate directories when you specify a hierarchy of directories. All the directories in the hierarchy must be empty.

Getting More Information

To display the man page and learn more about rmdir and the options available, type man rmdir in Terminal. There is one option that can be useful in certain situations.

You can remove several empty directories in one command. Just separate their names with single spaces, specifying either just their names (for relative paths, within the current working directory) or their paths (for absolute paths). Here's an example:

```
$ rmdir Files ~/Documents/OldDocuments ../Video
```

You can also use `rmdir` to remove a hierarchy of directories, as long as all the directories are empty. Use the `-p` option as follows:

```
$ rmdir -p Directory1/Directory2/Directory3
```

One disadvantage to using the `rmdir` command is that, unlike the `rm` command, it has no `-i` (interactive) option, which asks you to confirm the deletion, and no `-v` (verbose) option ; these deficiencies limit its value.

Removing Files with *rm*

The `rm` (**rem**ove) command is one of the most powerful and dangerous commands you can use in Terminal. Be forewarned: when you remove a file with `rm`, it is deleted *forever*. While some file recovery programs may be able to find files deleted in this manner, it is safest to assume that what is eliminated with `rm` is gone for good.

While many Unix commands are safe to run, even if you have little experience, `rm` is like a loaded gun. For this reason, you should use this command with the utmost care. However, there is a simple way to apply a safeguard to this command (and others); see the next section for a safety measure that will take the worry away.

Running the `rm` command is relatively simple. Look at the contents of this directory:

```
$ ls
File1 File2 File3
```

To remove one of these files, run the following:

```
$ rm File1
```

You can check to make sure it worked:

```
$ ls
File2 File3
```

You can see that the file you removed is indeed gone; it no longer shows up in the list.

A Safer Way to Remove Files

The first way of removing files, as shown above, is really for those people who are totally confident with the command line. It's working without a net, though; as I said above, `rm` is very powerful, but there is a safeguard you can use to protect yourself. The `rm` command has an option, `-i` (for interactive), which tells Terminal to ask you to confirm that you really want to delete each file.

To use this option, run the command as follows:

```
$ rm -i File2
```

Terminal will ask you to confirm.

```
remove File2?
```

Type `y` for yes or `n` for no.

```
remove File2? y
```

If you type **y**, the file will be deleted. If you type **n**, or press Return, the file will not be touched. In both cases, Terminal will display a new prompt; it gives no other information, and you need to run the `ls` command again to see what's in the directory.

A good thing to do is set `rm -i` as an alias for `rm`, so you can type `rm` and use the interactive option without needing to remember it or typing those extra characters. See Chapter 16, "Configuring the Shell," for an explanation of how to do this.

An Even Safer Way to Remove Files

There is one good thing about deleting files in Mac OS X (and in previous versions of Mac OS): file deletion is always a two-step procedure. You first move files to the Trash, then you delete them (unless you are deleting files on a remote volume; they are deleted immediately when you move them to the trash).

With the command line, this is much more radical. You remove files, and they are gone.

But there's a safeguard you can use, combining the best of both worlds. Instead of removing files with the `rm` command, you can use the `mv` command to move them to the Trash. This lets you work with a safety net to make sure you haven't goofed and deleted the wrong file.

To do this, use the following command:

```
$ mv filename ~/.Trash
```

If you look in the Trash, you'll see the file you moved. You can later empty the Trash when you are sure you want to delete the file.

There is one thing to note, however. Unlike when you drag or send files to the Trash in the Finder, the above command replaces any like-named file that is in the Trash. So make sure that you don't want to get back other files that are already in the Trash and that may have the same name.

You can also create your own command alias to do this more easily. See Chapter 16 for instructions on creating this alias.

Getting More Feedback with *rm*

The `rm` command has another option, `-v` (verbose), which shows the names of files as it removes them. If you use this option, Terminal will show the following:

```
$ rm -v File3
File3
$
```

I have added the prompt after the file removal to illustrate how this is displayed in Terminal. All this option does is show the name of the file, but it can serve to confirm exactly which file has been removed. Of course, if you made a typo and removed the wrong file, it's too late now!

Removing Directories with *rm*

While there is a special command for removing empty directories (`rmdir`; see above), the `rm` command also lets you do this through the `-d` option. I cannot repeat often enough how dangerous this command is. Don't forget to check the "A Safer Way to Remove Files" section above to see the best safeguard to protect against accidental deletions.

To remove a directory, use the following:

```
$ rm -d Directory1
```

You can use the `-i` and `-v` options, as mentioned above, for additional security:

```
$ rm -div Directory1
```

Hands On: Shredding Files with *rm*

There are many programs on the market that shred files: what they do is not only delete files but overwrite the disk space they used to make sure that file recovery programs cannot get them back. This is a useful security measure for many people, and it's no surprise that there is a Unix command that can do this without the need for additional software.

The trick is to use the `-P` option with `rm`. This overwrites files first with the byte pattern `0xff`, then `0x00`, and then `0xff` again; this triple protection may seem overkill, but if you are really concerned about shredding your files it's reassuring to know that you're covering all bases.

The command is run as follows:

```
$ rm -P fileName (fileName2 ...)
```

Naturally, you can also use the `-i` option, presented above, to confirm removal of the file(s), or the `-v` option to show the names of the files being removed.

Note that the `rm` command's three-pass deletion is good, but not the best way of shredding your files. The Finder offers a U.S. Department of Defense–approved system of deletion offering seven passes, which is guaranteed to prevent recovery. Some companies or government agencies may need to use this system, and in such cases, the `rm` command's secure deletion is insufficient.

The `rm` command can also remove directories recursively, using the `-r` (recursive) option, deleting a directory and all its subdirectories, as well as any files they contain. This is like dragging a folder that contains subfolders and files to the Trash in the Finder, except you can't drag it back out of the Trash if you want to keep it.

Let's look at how this works, and how you can use the `-i` option for minimal protection.

First, create a few nested directories:

```
$ mkdir Directory1 Directory1/Directory2 Directory1/Directory2/Directory3
```

ON THE COMMAND LINE

Overview of the *rm* Command

COMMAND SYNTAX

```
rm [option(s)] file1/directory1 file2/directory2 ...
```

FINDER EQUIVALENT

Move to Trash (though `rm` deletes the files or directories immediately).

OVERVIEW OF OPTIONS FOR *RM*

`rm`	This command removes (deletes) the file(s) and/or directory(ies) specified in its arguments. This deletion is quick and painless. It is also permanent: when removing files or directories, there is no way to get them back.
`-i`	This is the interactive option. The shell will ask you to confirm before removing a file that would overwrite an existing file. If you type `y`, the deletion will continue. If you type `n`, it will stop.
`-v`	This is the verbose option. It tells the shell to display the name(s) of files or directories removed after running the command.
`-d`	This option removes directories.
`-P`	This option shreds files by overwriting them. This overwrites files first with the byte pattern 0xff, then 0x00, and then 0xff again.
`-r (or -R)`	This is the recursive option. It removes directories and files recursively, starting from the directory specified, it removes every subdirectory and file within that directory.

GETTING MORE INFORMATION

To display the man page and learn more about `rm` and the options available, type `man rm` in Terminal. Several additional options can be useful in certain situations.

Then, to remove all three of these directories, run the following:

```
$ rm -ir Directory1
remove Directory1? y
remove Directory1/Directory2? y
remove Directory1/Directory2/Directory3? y
```

Obviously, if you don't use the `-i` option, the command will just remove all the directories without any feedback.

But what if you decide that you don't want to remove one of the directories? You can type `n` at any point to keep it. Watch what happens then:

```
$ rm -ir Directory1
remove Directory1? y
```

```
remove Directory1/Directory2? y
remove Directory1/Directory2/Directory3? n
rm: Directory1/Directory2: Directory not empty
rm: Directory1: Directory not empty
```

If at any point in the hierarchy you don't want to delete a file or directory, the system cannot let you keep an item without its parent directory. In the above example, the decision to not remove `Directory3` means that `Directory2` (its parent directory) could not be deleted; `Directory1` (the parent of `Directory2`) could also not be deleted, so none of these three directories will be removed. However, if they contain files, and you answer `y` to the `rm` command's confirmation requests for these files, they will be removed.

Aliases and Links

When the Finder sees an alias or a symbolic link, it treats them both the same—they will be displayed the same, and double-clicking on either one will take you to its source (if a folder) or will open the file or application. Prior to Mac OS X 10.2, aliases maintained references to files or folders through their unique identity first and their filepath second. Beginning with Mac OS X 10.2, the filepath was given priority. This means that from 10.2 on, you can delete the original file or folder and create a new one with the same name and the alias will still work. The advantage to aliases is that the Finder can keep track of the original if it is moved.

Can't Empty the Trash?

While in the Finder, you'll occasionally find that you cannot fully empty the Trash. There may be one or more files that you cannot empty because of their permissions, or because they are locked. The `rm` command lets you empty the Trash easily.

The first thing you need to know is that there are several Trashes. Each user has their own `Trash` directory (`~/.Trash`) and each volume has one (`/.Trashes`). So, if you have several volumes, you'll have one `Trash` directory to empty for your user account and one per volume. Run these commands to empty all the Trash:

```
$ sudo rm -rfi ~/.Trash/*
$ sudo rm -rfi /.Trashes/*
$ sudo rm -rfi /Volumes/<volumename>/.Trashes/*
```

The first command empties your user Trash, the second the startup volume's Trash, and the final one empties the trash for any other volume (enter the name of the volume where the command shows <*volumename*>.

Since these commands use the `-i` (interactive) option, Terminal will ask you to confirm deletion of each file. If you're sure you want to delete all these files, you can leave this option out.

The second and third commands empty the Trash for *all users*. So only use them if you really need to do this. If you just want to empty the Trash for your user account, run these commands:

```
$ sudo rm -rf /.Trashes/`id -u`/*
$ sudo rm -rf /Volumes/*/.Trashes/`id -u`/*
```

The above commands get your user ID number and delete the contents of the appropriate Trash subdirectories.

Symbolic links work strictly according to filepaths. If you move the original, the symbolic link is broken. But only symbolic links work in Terminal—you cannot do anything with aliases from the command line.

Creating Symbolic Links with *ln*

The `ln` (**lin**k) command creates links from Terminal. As mentioned above, these symbolic links function like Finder aliases (though if the original is moved, the link breaks). Its syntax is very similar to that of the `cp` or `mv` commands: you type the command followed by a source and target path.

```
$ ln -s /Volumes/Files/Current ~/Current
```

In the above example, I created a link for the `Current` directory, located in the `Files` partition, and placed it in my home directory. Inspecting this directory shows the following:

```
$ ls -F
Current@  Documents/  Movies/  Pictures/  Sites/
Desktop/  Library/    Music/   Public/
```

Remember, `ls -F` shows the directory contents in short form, but adds a slash after a directory and an `@` after a symbolic link.

You can also run `ls -l` to get more information:

```
$ ls -l
total 0
lrwxr-xr-x    1 kirk   staff     33 Nov 21 09:42 Current -> /Volumes/Files/Current
drwxr-xr-x    7 kirk   staff    238 Nov 20 22:24 Desktop
drwxrwxrwx   17 kirk   staff    578 Nov 20 22:23 Documents
drwxr-xr-x   43 kirk   staff   1462 Nov 20 21:17 Library
drwxr-xr-x    2 kirk   staff     68 Jan 10  2002 Movies
drwxr-xr-x    5 kirk   staff    170 Sep 24 01:02 Music
drwxr-xr-x   10 kirk   staff    340 Jun 25 19:26 Pictures
drwxr-xr-x    7 kirk   staff    238 Nov 20 22:12 Public
drwxr-xr-x    4 kirk   staff    136 May 28 16:56 Sites
```

As you can see, `Current` points (`->`) to `/Volumes/Files/Current` and the entry type flag is `l`, showing that it is a link.

Now that you have a symbolic link in this directory, you can use `cd` to move into it:

```
Walden:~ kirk$ cd Current/
Walden:~/Current kirk$
```

The prompt treats the symbolic link like a directory—it does not show the actual path. But if you run the `pwd` command, you will see where you really are:

```
$ pwd
/Volumes/Files/Current
```

whereas running `pwd -L` shows the logical path to the directory, or how you got there:

```
$ pwd -L
/Users/kirk/Current
```

(For more on `pwd`, see Chapter 4, "Navigating the File System.")

Absolute versus Relative Paths for Symbolic Links

Unlike Finder aliases, symbolic links are not updated when you move their targets in your file system. If you create an alias to a file using the Finder and move the file somewhere else on the same volume, the Finder will still know where it is and be able to open it. (The Finder won't follow aliases copied to other volumes, because when moving a file to another volume you copy it. It only follows the original file the alias points to.) If you copy the alias somewhere else (even to another volume), it will still find the original file.

With symbolic links created in Terminal, the situation is a bit different. If you create a symbolic link then move the original, the link breaks. But if you create a new file with the same name as the original, in the original location, the symbolic link will work again.

If you move the symbolic link, there are two possibilities for what will occur: if you specified a relative filepath as the source, the link may break. But if you used an absolute filepath when creating the link, it will still work. In this example:

```
$ ln -s Current ~/Current
```

I created a link with an absolute filepath. I told the shell to make a link to `/Volumes/Files/Current` and place it in `~/Current`. If I were in my home directory and ran the following command:

```
$ ln -s /Volumes/Files/Current ~/Current
```

it would have the same effect, at least at first. I would have a link in my home directory that points to `/Volumes/Files/Current`. But since this link has a relative filepath, it would break if I decided to move it anywhere else.

Aliases and the Command Line

Most Mac users are familiar with aliases—they allow you to create virtual pointers to files or folders that can be placed anywhere on your computer. When you double-click an alias to an application, the application opens; when you double-click a file alias, that file opens; and when you double-click a folder alias, a window opens displaying the contents of that folder.

Aliases are very practical on the Mac for organizing frequently used files, folders, and applications in different locations. You can put aliases to all your common applications in one folder, and, instead of opening lots of windows to get to the applications, can open them easily with just a double-click. You can also use file and folder aliases to group pointers to common files and folders, again saving time.

Continued on next page

Aliases and the Command Line *(continued)*

Unix systems don't grok Mac OS aliases. (But note that an *alias* under Unix has a totally different meaning. See Chapter 16 for information on shell aliases.) Unix systems have an equivalent to Mac OS aliases called *symbolic links*. These are pointers to files, directories, or applications that work from the command line almost the same as aliases work in the Finder.

If you try to use cd to move into a directory alias, you will see the following:

```
$ cd Current
Current: Not a directory.
```

When running `ls` to look at the contents of a directory, there is nothing to indicate that `Current` is an alias:

```
$ ls
Current      Library     Pictures
Desktop      Movies      Public
Documents    Music       Sites
```

But if you run `ls -l` you can see the difference:

```
$ ls -l
total 320
-rw-r--r--   1 kirk  staff     0 21 Nov 10:06 Current
drwx------  17 kirk  staff   578  8 May 16:29 Desktop
drwx------  36 kirk  staff  1224  6 May 08:56 Documents
drwx------  67 kirk  staff  2278 29 Apr 23:01 Library
drwx------   6 kirk  staff   204  5 Feb 20:47 Movies
drwx------  20 kirk  staff   680 21 Mar 21:34 Music
drwx------  10 kirk  staff   340 29 Apr 23:22 Pictures
drwxr-xr-x   8 kirk  staff   272  6 May 10:59 Public
drwxr-xr-x   8 kirk  staff   272 24 Feb 16:15 Sites
```

Terminal sees `Current`, the alias to another folder, as a kind of file. Note the flags at the left of this line:

```
-rw-r--r--   1 kirk  staff     0 21 Nov 10:06 Current
```

The first flag, the entry type flag, shows whether the item is a directory (the d flag), a symbolic link (the `l` flag), or one of several other types of special items. Since this flag is blank, Terminal sees it as just another file.

Aliases, Symbolic Links, and Hard Links

Symbolic links are very similar to Finder aliases though they react differently when moved, depending on whether you used absolute or relative filepaths when creating them.

Unix systems offer another kind of link, called *hard links*. A hard link is more than an alias, and different from a symbolic link, because it links directly to the original file's location in the file

system. When you delete a hard link, the target remains in its original location. When you delete the target of a hard link, the hard link remains, with the content of the file.

An example may help you clarify this. Imagine a puppet held by several strings to its frame (the wooden piece that the puppeteer holds). If hard links were strings, you could cut one, two, even three of these strings (depending on how many there are) and, as long as there is still one string holding on to the puppet, it remains attached to the frame. Cut the last string and the puppet falls.

Filenames in the file system are actually hard links. Each bit of data that makes up a file is referenced by a filename. When you move a file, you aren't moving its data, just its name within the file system tree. When you delete a file, you delete the reference to that data, not the data itself.

For all intents and purposes, a hard link is the exact same thing as its target file; unlike symbolic links or aliases, deleting the target does not actually delete the file. It is more like giving a file several names. As long as there is at least one hard link to a file, the file system maintains the file and only deletes the file when there are no more hard links to it. In addition, you can move a hard link or its target and the link will still work, and if you move a hard link to another volume, it becomes a file—no longer linked to the original target, it is exactly the same as the original file and can be edited accordingly.

If you are used to working with aliases, you may find the concept of hard links a bit confusing. The fact that you can delete the original file, yet not actually delete it, may lead to confusion. The other problem with hard links is that they cannot refer to directories and cannot be used to refer to files on other volumes, whether on the same computer or a different computer on a network. In most cases, symbolic links will be sufficient.

As you can see in Table 5.2, aliases, symbolic links, and hard links have different properties and different usages. The one to use depends on how you need to work with it.

When looking at links in Terminal, you can tell which type they are by using the `ls -l` command. Here are three files: a normal file, a hard link, and a symbolic link.

```
-rw-r--r--    2 kirk     staff    0 Jan 13 10:51 testfile
-rw-r--r--    2 kirk     staff    0 Jan 13 10:51 hardlink
lrwxr-xr-x    1 kirk     staff    8 Jan 13 10:51 symlink -> testfile
```

The first file is a normal file, but you can tell by looking in the second column that a hard link to this file exists. If the file had no hard links, the directory link count in this column would be 1; since it is 2, you know there is a hard link, though you don't know where.

The second file, a hard link to the first file, shows the same information except for its name. The only way you know it was created as a hard link is because of its name.

The third file is a symbolic link, and this shows in two ways. First, the file mode section at the beginning of the line starts with the letter `l`, indicating this is a symbolic link. Second, the filename shows both the name of the link and the file it points to, with the `->` symbol between them. Also, the size of this file is equal to the length of the file it links to; since `testfile` is 8 letters, `symlink` has a size of 8 bytes.

TABLE 5.2: PROPERTIES AND USAGE OF ALIASES, SYMBOLIC LINKS, AND HARD LINKS

Finder aliases	Symbolic links	Hard links
Aliases can link to files, directories, or applications.	Symbolic links can link to files or directories. They can link to command-line applications as well, and when created to Finder applications, appear and work like Finder aliases.	Hard links can only link to files. (The only exceptions are the `.` and `..` hard links, which are created with each new directory, but only the `mkdir` command can create these links.)
The Finder resolves aliases when files are dragged onto them.	The Finder resolves symbolic links when files are dragged onto them.	You cannot create hard links to directories or applications, so there is no reason to drag items onto them.
The Finder resolves aliases when you double-click them.	The Finder resolves symbolic links when you double-click them.	The Finder resolves hard links to files when you double-click them.
Aliases can refer to items across volumes.	Symbolic links can refer to items across volumes.	Hard links cannot refer to items across volumes.
Aliases can be moved and still work. The targets of aliases can also be moved and still work.	Symbolic links can be moved and still work (if absolute filepaths are used). If you use a relative filepath, the link breaks when moved. (However, a relative link using `../` at the beginning of its path will still work if it is moved to another directory that is one level below its target.)	Hard links can be moved on the same volume and still work, whether you use an absolute or relative filepath when creating them. If moved to another volume, they become copies of the original files.
Aliases give visual indications in the Finder (the alias icon has an arrow), and the Get Info window shows their paths. They also maintain any custom or application icons the item has.	Symbolic links give visual indications in the Finder (the symbolic link icon has an arrow), and the Get Info window shows their paths. They also maintain any custom or application icons the item has.	Hard links are blank file icons, showing nothing about what they are. They do not take on specific custom or application icons.
Aliases give no indication of what they are in Terminal.	Symbolic links show clearly in Terminal; a link name is shown like this: *link_name* `->` *original_file*.	Hard links give no indication of what they are in Terminal, but they are equivalent to their targets.
You cannot follow aliases in Terminal.	You can open (if links to files) or follow (if links to directories) symbolic links from Terminal.	You can open hard links (or, more correctly, the files they point to) from Terminal.

Hands On: Using Links to Work with the Developer Tools

If you have installed the Developer Tools provided with Mac OS X, you probably want to access some of their special commands, such as `MvMac` and `CpMac`. (For more on the Developer Tools, see Interlude 8.) Since these commands are located in `/Developer/Tools`, you cannot invoke them without specifying their entire path, unless you add this path to your shell's `PATH` variable. (See Chapter 16 for more on the `PATH` variable.)

Another way to do this is to create symbolic links to the commands you use often and put them into `/usr/local/bin`, where the shell looks automatically. This saves you from changing your `PATH` variable, but it also ensures that the commands remain where they belong. Though you could physically move these commands, they would be in the wrong place when you install the next Developer Tools upgrade.

To create these links, run the following commands:

```
$ sudo ln -s /Developer/Tools/CpMac /usr/local/bin/cpmac
$ sudo ln -s /Developer/Tools/MvMac /usr/local/bin/mvmac
```

If you run the commands as above, you'll do something else that will make using these Developer Tools commands a bit easier: by naming the links in lowercase letters, you won't need to remember that the original commands actually contain a combination of both uppercase and lowercase. (These names are case-sensitive.)

Once you have created these links, you can use these commands by simply typing their names instead of their full paths. If you use `bash`, run the `hash -r` command; if you use `tcsh`, run the `rehash` command so the system finds the new links in `/usr/local/bin` and you can use them.

On The Command Line

Overview of the *ln* Command

Command Syntax

```
ln [option(s)] source [target]
```

Finder Equivalent

Make Alias.

Overview of Options for *ln*

- `ln` This command creates a link between a source (original) file or directory and a target file or directory. If you don't use the `-s` option, this is a hard link.
- `-s` This option creates a symbolic link.

Getting More Information

To display the man page and learn more about `ln`, and the options available, type man `ln` in Terminal.

Copying Directories with *ditto*

As we saw earlier, the `cp` command copies both files and directories but has one important disadvantage: it does not copy resources and certain other file information that is required by the Finder. Because of this, Apple has provided a special command-line tool with its Developer Tools: `CpMac` copies files so all Finder information is maintained. Together with `MvMac`, which is an enhanced version of the `mv` command, these two tools offer "safe" copying of files and applications. (For more on using these tools, see Interlude 8.)

There is another program included in the basic OS X installation for copying directories and their contents: `ditto`. This command is most useful for two things:

- Copying Mac OS X resources and metadata. The `ditto` command correctly copies Mac OS X applications, which are stored as a kind of folder, and also copies other metadata that the Finder uses.
- Backing up data. The `ditto` command can back up your data safely, keeping all attributes the Finder needs, and can also make a bootable clone of your startup volume.

This command works in two ways. In the first form, you can copy a directory to another location, specifying the source directory and the name of the new target directory. You can run the command like this:

```
$ ditto directory1 directory2
```

The command creates the new directory and copies the source directory to the destination, which can have either the same name as the source (if in a different parent directory) or a different name.

To use `ditto` to copy Mac OS X resources and metadata (this includes Mac OS aliases and custom icons), you must use the `-rsrcFork` option. If the destination is on a file system that does not support resource forks, `ditto` will store this data in AppleDouble files.

```
$ ditto -rsrcFork directory1 directory2
```

If you specify the name of the destination directory and it does not exist, `ditto` creates it. In the following command, this works more like the `cp` command:

```
$ ditto -rsrcFork ~/Documents ~/Desktop/DocumentsBackup
```

As you can see, I copied my `Documents` directory to the desktop. Since I specified the name of the destination (`DocumentsBackup`), `ditto` created this directory, then copied the contents of my `Documents` directory into it.

But `ditto` becomes very interesting when copying into an existing directory. In this case, the `ditto` command copies the *contents* of the source directory into the destination directory. This is very different from how `cp` works. As the `ditto` man page says, "If the destination directory does not exist it will be created before the first source is copied. If the destination directory already exists then the source directories are merged with the previous contents of the destination."

When copying a directory with cp, the source directory and its contents are copied to the destination directory. If you have a directory called dailyFiles that contains seven files (Monday, Tuesday, etc.) and you run the following command:

```
$ cp -R dailyFiles Backup
```

you end up with the following structure:

```
$ ls Backup
dailyFiles
```

The dailyFiles directory was copied to the Backup directory.

But when doing this:

```
$ ditto dailyFiles Backup
```

the result is as follows:

```
$ ls Backup
Friday  Monday  Saturday  Sunday  Thursday  Tuesday  Wednesday
```

As you can see, ditto did not copy the dailyFiles directory to Backup, but only its contents.

This difference is both subtle and tricky. On the one hand, it allows you to copy or back up the contents of a directory or volume to another directory or volume, maintaining the exact same structure as the original, without replacing the parent directory. If you back up your entire Users directory to a Backup volume, for example, you will not see a Users directory after the backup, but rather a group of directories with the names of each user. The structure is that which was *inside* the source directory.

But this is also tricky. You are probably used to copying a directory and finding that directory inside the destination. So you need to make sure that if you want to back up all your user folders, you run ditto with the Users directory, not each individual user directory, as the source.

Finally, ditto overwrites all existing files, symbolic links, and directories in the destination, but ensures that all these items have the same mode, owner, and group as the source items. Files cannot overwrite directories or vice versa, so if you have a directory with the name of a file, it will not be copied.

Copying Applications with *ditto*

As I mentioned earlier, only ditto and CpMac can copy applications correctly under Mac OS X. If you haven't installed the Developer Tools, ditto is the only way you can do this.

When copying items with ditto, remember that ditto copies the contents of a directory. You may also recall that Mac OS X applications are actually directories masquerading as single items. So, with this in mind, you can probably imagine what will happen if you try to copy an application with ditto.

```
$ sudo ditto -rsrcFork /Applications/Calculator.app ~
```

(Notice the use of sudo before this command—in many cases, you will get a "Permission denied" result from Terminal if you run ditto on an application without sudo. For more on using sudo, see Interlude 7, "Using *sudo*.")

In the above example, I tried to copy the Calculator application to my home directory. But when I look in my home directory, I see the following:

```
$ ls ~
Contents   Desktop  Library   Music   Public
Documents  Movies   Pictures  Sites
```

There's nothing called Calculator, but there is something called `Contents`. If you recall from the section on the `cp` command, copying an application has surprising consequences (at least copying native Mac OS X applications, which are, in fact, directories containing other files and directories). The `cp` command strips the resources and metadata that the application needs to appear as an application, leaving behind nothing but a directory called `Contents`. But `ditto`, with the `-rsrcFork` option, is supposed to keep this data, right?

Well, it does, but you need to run `ditto` differently. Instead of specifying a source (the application) and a destination (the directory to which you want to copy the application), you need to run `ditto` like this:

```
$ ditto -rsrcFork /Applications/Calculator.app ~/Calculator.app
```

When done in this manner, `ditto` copies the application as is, and it works fine.

One final note: I have used the `-rsrcFork` option here, but if you run the above command without it, it will work just fine. Most of Apple's own applications (such as Calculator) have no resource forks, but other applications may have them. It is safer to use this option, though if you copy an application without it you'll probably find out right away whether it works or not.

Backing Up Your Files with *ditto*

As we saw above, `ditto` is the safest way to copy Mac OS X files and applications. If you want to regularly back up your files, you can use `ditto` to do this.

Start by choosing a location for your backups. The best way is to have a separate partition or hard disk for backups. But if you haven't partitioned your hard drive, you could also use an external medium such as a Zip drive.

To use a removable medium such as a Zip drive, insert your Zip cartridge and run the following commands:

```
$ cd /Volumes
$ ls
```

The first command moves you to the `Volumes` directory, which contains the names of all mounted volumes. The second lists these volumes. Use the exact name of the Zip cartridge or other medium, as shown here. When I insert a 250MB Zip cartridge in my drive, it shows up as ZIP 250.

If you have a partition dedicated to backups, you can copy everything to this location. If not, create a directory on this partition with the same name as your user name. (I'll show the following commands with my user name, kirk; replace this with your user name.)

```
$ cd /Volumes/'Zip 250'
$ mkdir kirk
```

WHY USE *DITTO*?

The `ditto` command is the command-line equivalent of a Finder drag-and-drop copy. So why use `ditto` instead of copying in the Finder? The main reason to use this command is so you can add it to scripts, or have it run automatically using `cron`. (For more on `cron`, see Interlude 9, "Automating Commands.") Otherwise, unless you cannot access the Finder (because of problems on a computer, or because you have logged in remotely) there is little reason to use it. Drag-and-drop copies are generally faster and easier than command-line copies.

However, there is one thing you can do with `ditto` that you cannot do from the Finder: you can use it to create a bootable backup of your startup volume, and one reason you would want to do this is to copy invisible files. For a complete explanation on cloning your startup volume, see Interlude 4, "Cloning your Mac OS X Startup Volume."

If you are backing up your home directory, first make sure there is enough room on the destination; if there isn't, you will have to back up something that will fit—your `Documents` directory, for example. Run the following command, changing *partitionName* to the name of your partition or external backup medium:

```
$ ditto -rsrcFork /Users/kirk /Volumes/partitionName/kirk
```

If I run this command to my backup partition (called `Backup`), it would be as follows:

```
$ ditto -rsrcFork /Users/kirk /Volumes/Backup/kirk
```

If I run this command to my Zip cartridge, it would be as follows (note the use of quotes, because of the space in the name of the Zip cartridge):

```
$ ditto -rsrcFork /Users/kirk /Volumes/'Zip 250'/kirk
```

Backup may take a few minutes, depending on how many files you have in your directories. The copy will be finished when Terminal displays a prompt.

When running this command, you may see some messages like this:

```
/Users/kirk/Library/Preferences/com.apple.loginwindow.plist:
Permission denied
```

This means that there is an issue with permissions that you don't have. These files are not copied, but the rest of your files and directories are copied. (To copy *every* file, you can run the command using the `sudo` command. For more on `sudo`, see Interlude 7.)

ON THE COMMAND LINE

Overview of the *ditto* Command

Command Syntax

```
ditto [option(s)] [source] [destination]
```

Finder Equivalent

Copy.

Overview of Options for *ditto*

`ditto`	This command copies the source argument to the destination. If the destination directory does not exist, it will be created before the copy is made. If it does exist, the contents of the source are merged in the destination directory.
`-rsrcFork`	This is the resource fork option. It tells `ditto` to copy Mac OS resource forks and metadata.

Getting More Information

To display the man page and learn more about `ditto`, and the options available, type `man ditto` in Terminal. This command has several options that can be useful in certain situations.

Hands On: Backing Up Files across a Network

The `ditto` command is a great tool for backing up files locally, but if you want to back up files and synchronize directories across a network, the `rsync` command is what you need. This command can set up a secure connection and copy files and directories to another computer, maintaining an exact copy of the source directory on both sides—it can delete any files in the destination that are not in the source.

For more on the `rsync` command, see Chapter 13, "Using the Network."

Summing Up

This chapter has shown you the essential commands for copying, moving, and deleting files and directories, as well as creating directories. These commands—`cp`, `mv`, `rm`, `mkdir`, `rmdir`, and `ditto`—do similar things as drag-and-drop copying in the Finder but, as some of the examples show, offer more power and flexibility. While the Finder remains easier to use for most operations, these commands give you a powerful alternative and in some cases provide tools that the Finder cannot offer.

Interlude 4: Cloning Your Mac OS X Startup Volume

While backing up your personal files is essential to make sure you don't lose any data in case of a crash or hardware problem, backing up your startup volume is purely optional, yet can be useful as insurance. If you do this from time to time, such as before making any new system upgrades, you can always revert to a working system if you encounter any problems. It can take a long time to reinstall the system and all its upgrades if you do have problems; a clone of your startup volume lets you get up and running in a short time.

Under previous versions of Mac OS, you could merely copy your System Folder to another volume to make a backup and recopy it to your startup volume in case of problems. You could even boot off this backup easily, as long as your backup System Folder was *blessed*, or set to be bootable (usually, it was sufficient to open the System Folder for this to occur).

Under Mac OS X, you cannot just copy your startup volume to another disk. Well, you can, but you won't be able to boot from this backup. There are many files that don't get copied if you do a drag-and-drop copy, and many of the Unix permissions and links are either changed or damaged during copy. To make a useful copy of your Mac OS X startup volume, you must *clone* it, making an exact copy not only of all the files, but also of the permissions and settings.

While you can use third-party tools to clone your Mac OS X startup volume (using a program such as Mike Bombich's shareware Carbon Copy Cloner, `http://www.bombich.com/software/ccc.html`, or Intego Personal Backup X, `http://www.intego.com`), you can also accomplish this task using the command line.

Note: this procedure covers Mac OS X 10.3 (Panther) and 10.2 (Jaguar) and does not work with older versions of Mac OS X.

Considerations for Cloning a Mac OS X Startup Volume

Several issues must be considered when cloning a Mac OS X startup volume. The following is adapted from Mike Bombich's "Guide to Backing Up Mac OS X," found at `http://www.bombich.com/mactips/image.html`.

File Permissions Must Be Preserved Many files belong to the root user, so you cannot simply copy these files from the Finder. There are other issues with permissions, such as the `setuid` bit, a feature of a file that, when executed, gives the file or application the same privileges as the owner of the file; if the owner of the file is root, then root privileges are granted during the execution of this file. Copying via the Finder sets the owner of the new files to the user who copied them and assigns a default set of permissions. Many applications and system files will not work properly with the default Finder settings.

The Invisible Unix System Files Must Be Copied Some of the essential directories for Mac OS X are invisible: these are `/private`, `/bin`, `/usr`, and `/sbin`. These directories hold critical files that allow

the computer to boot and operate. There are also other invisible files at the root level of the file system that the Finder cannot copy.

Unix-Style Links Must Be Preserved Symbolic links and hard links are different from Mac aliases, and the Finder does not copy them correctly. Because there are some critical symbolic links on a Mac OS X volume, the integrity of these files must be preserved when you clone the volume.

Special Directories Some directories are populated by the system. For example, the `Volumes` directory is populated with directories corresponding to the names of volumes you have on your system. These directories are called mountpoints and are created on the fly by Apple's `autodiskmount` utility. Because these directories do not contain data on your startup volume, they do not need to be copied during a clone operation. The `Volumes` directory is just a placeholder (and Mac OS X recreates the `Volumes` directory on startup). The `/dev` directory is also a placeholder, for system devices such as disk drives, output devices, and communications devices. The list of devices in this directory is created each time the computer is started up and when new hardware is added, so it is unnecessary to copy the items in this directory. Because this is a Unix system directory, however, you will not have a bootable volume unless this directory is recreated on the cloned disk. Creating an empty directory is sufficient. Likewise, it is important to back up `mach_kernel` (the most important file in the system), but `mach` and `mach.sym` are destroyed and recreated each boot by the `/etc/rc` boot script. Finally, the `Network` folder at the root level does not need to be backed up because it is populated by the system on startup.

Resource Forks Must Be Preserved While Apple is trying to move away from resource forks, many applications and documents still use them. Because of this, any backup or cloning utility must preserve the resource forks. If you try to clone a Mac OS X disk without preserving resource forks, many of your personal documents will be damaged.

Preparing to Clone a Mac OS X Startup Volume

Do the following before cloning your Mac OS X startup volume:

- Make sure the Ignore Ownership on This Volume setting is not checked for your target volume. To check this setting, click on the target volume's icon in the Finder, select File ➢ Get Info, then click the disclosure triangle next to Ownership and Permissions. Make sure the box at the bottom is *not* checked, otherwise permissions and ownership settings will not be preserved, no matter how you copy files.
- Run Disk Utility on the target and source volumes before cloning. This is not required, but is a good idea to avoid disk- or directory-related problems during cloning. If you are cloning to an external FireWire device, it's a good idea to reformat (not simply erase) the drive with Disk Utility prior to cloning.

Cloning a Mac OS X Startup Volume with *ditto*

The `ditto` command preserves permissions when run as root and preserves resource forks when run with the `-rsrcFork` option. (For more on `ditto`, see Chapter 5, "Working with Files and Directories.") This command is easy to use to clone a Mac OS X startup volume, and you can clone a disk with just a few steps. Here's what you need to do to clone your startup volume. (Note: use the following procedure at

your own risk. Make sure, before erasing your original volume, that you are able to boot from the clone and that no files are missing.)

In the following procedure, the volume used for the clone is called `/Volumes/Backup`. Change this to reflect the name of the actual volume you are using. Also, you must have an administrator password to run these commands. Terminal will ask you to enter that password after you type the first command.

Note that some of these commands may take a while to run, and the commands don't give you any feedback in Terminal. You may have a couple hundred megabytes of files in your `/Applications` folder, for example, and this takes a long time to copy.

1. Use `ditto` to copy each of the visible directories from your startup volume to your backup volume. You need to repeat this step for each of these files or directories at the root level of your drive:

```
$ sudo ditto -rsrcFork /Applications /Volumes/Backup/Applications
$ sudo ditto -rsrcFork /Developer /Volumes/Backup/Developer
$ sudo ditto -rsrcFork /Library /Volumes/Backup/Library
$ sudo ditto -rsrcFork /System /Volumes/Backup/System
$ sudo ditto -rsrcFork /Users /Volumes/Backup/Users
$ sudo ditto -rsrcFork /System\ Folder /Volumes/Backup/System\ Folder
```

If you have not installed the Mac OS X Developer Tools, you won't have a `Developer` directory; if you haven't installed Mac OS 9 on the same volume as Mac OS X, you won't have a System Folder.

If you have installed Mac OS 9, you'll also want to copy the Mac OS 9 `Applications` folder (the following command should all be on one line):

```
% sudo ditto -rsrcFork '/Applications (Mac OS 9)'
     '/Volumes/Backup/Applications (Mac OS 9)'
```

You don't need to back up any of these files or directories at the root level of your file system:

```
dev
Volumes
Network
etc
tmp
var
automount
.vol
mach
mach.sym
.DS_Store
Cleanup At Startup
```

```
TheVolumeSettingsFolder
File Transfer Folder
Trash
.Trashes
TheFindByContentFolder
```

However, if you find any other files or directories there, you should copy them. Use the same syntax as above to copy these additional items.

2. Use `ditto` to copy your system files:

```
$ sudo ditto -rsrcFork /cores /Volumes/Backup/cores
$ sudo ditto -rsrcFork /private /Volumes/Backup/private
$ sudo ditto -rsrcFork /usr /Volumes/Backup/usr
$ sudo ditto -rsrcFork /bin /Volumes/Backup/bin
$ sudo ditto -rsrcFork /sbin /Volumes/Backup/sbin
$ sudo ditto -rsrcFork /mach_kernel /Volumes/Backup/mach_kernel
$ sudo ditto -rsrcFork /.hidden /Volumes/Backup/.hidden
```

3. Recreate symbolic links and empty directories:

```
$ cd /Volumes/Backup
$ ln -s private/etc etc
$ ln -s private/var var
$ ln -s private/tmp tmp
$ mkdir dev Volumes Network
```

4. Bless the system (Mac OS X) and System Folder (Mac OS 9), if copied, on the target:

```
$ sudo bless -folder /Volumes/Backup/System/Library/CoreServices
$ sudo bless -folder9 /Volumes/Backup/System\ Folder -bootBlocks
```

The last step is required if you want to be able to boot from your clone. Another way to accomplish this is to select it as the startup disk in the Startup Disk pane of the System Preferences.

You should now have a bootable clone of your Mac OS X startup volume. To check that it works, select this volume as the startup disk in the Startup Disk pane of the System Preferences and restart. If all went correctly, you'll be able to start up from this volume.

If you ever need to copy this clone back to your startup volume, repeat the procedure using the volume containing the clone as the source and the desired startup volume as the destination.