1 Getting Started with Unix

To start you on your journey through Unix, we'll take a quick look at a few basic concepts and commands. In this chapter, we'll get you started with basic Unix skills, such as accessing a Unix account, logging in, and listing and viewing files and directories, among other things. We'll also show you how to explore Unix, see its capabilities, and discover just what you can do with it.

Chapter Contents

This chapter is essential for all Unix guru-wannabes. If you're a Unix novice, you should start at the beginning of this chapter and work through each section in sequence. With these basic skills mastered, you can then skip through this book and learn new skills that look useful or interesting to you. If you've used Unix before, you might peruse this chapter to review the basics and dust off any cobwebs you might have.

The skills covered in this chapter apply to any version of Unix you might be using, including Linux, Solaris, or BSD through your local Internet service provider (ISP); Solaris, AIX, Linux or HP-UX at work; your Mac OS X or Linux system at home; CygWin or Unix through VMware or Unix from a bootable CD on your home system; or any other *flavor* (that's the technical term) you can find. Keep in mind, though, that the exact output and prompts you see on the screen might differ slightly from what is illustrated in this book. The differences probably won't affect the steps you're completing, although you should be aware that differences could exist. (As much as possible, our examples will give you a sample of the diversity of Unix systems.)

Accessing a Unix System

Using a Unix system is different from working on a PC. Using a PC, the computer's hard drive is your personal space, and—generally—you don't have access to what's on someone else's hard drive. With Unix, you have your own personal space that's located within a much bigger system. You might think of Unix as an apartment building, with lots of individual apartment spaces, a central office, and perhaps other general spaces, like a maintenance office. With Unix, you have the entire system that houses dozens, hundreds, or even thousands of personal spaces as well as private spaces (for, say, the system administrator, bosses, or IT [Information Technology] department staff). You can access your apartment only, but the system administrator (or designated people with authorization) can access any apartment.

People choose to use Unix for a number of reasons:

- Control: Unix offers users more control and customization on the legal and licensing side as well as the "getting stuff done" side.
- Economy: Many flavors of Unix offer free or nearly free licensing.
- Power: Experienced Unix geeks can do more with less effort on Unix than Windows—for many things, at least.

In the final analysis, though, most Unix people end up sticking with Unix because they tried it, slogged through the initial learning curve, and then decided they like it.

Different types of Unix access

So, the first question is how you might access a Unix system to get started with all of this. Given that this is Unix, you have exactly 1.2 bazillion options. Let's look at these options:

- Connect to a shell account
- Access your company's (or school's or organization's) Unix system
- Use a live CD, such as an Ubuntu or OpenSolaris CD
- Do a Unix installation in a virtual machine on your computer
- Do a Unix-only installation on an old or spare computer
- Do a Unix/Windows installation on your everyday computer

Accessing a shell account

The traditional approach (back in the olden days, when we wrote the first version of this book) was to connect to a "shell account" provided by your dial-up ISP. That's still an option, if you have certain ISPs (and even with some broadband connections). If your ISP offers a shell account, go ahead and use it; it's still a good option. Try Googling "Unix shell account" as well.

Accessing your company's system

If not (that is, if you have a cable modem, DSL connection, or dial-up connection through any of the huge companies that provide Internet access, "not" is the case), you still have a ton of options. Check at work; many companies use Unix in a number of ways, and if you can provide the system administrator with appropriate quantities of cookies or other goodies, you may be able to get Unix system access.

Installing Unix on an old or spare computer

Alternatively, if you'd rather keep your Unix explorations closer to home, you can manage that as well. If you have an older computer sitting around (say, anything that's a Pentium III or later), you can just install Unix (Linux, Solaris, or whatever) on that, and likely without hassles or problems. You could make it work on even older computers, but given how cheap new and used computers are, it's likely not worth the trouble. Either way, you'll download a CD or DVD from the Web, burn it onto a disc, and boot your system with the disc in the drive. The installation will start, and a few questions and few minutes later, you'll be all set.

Installing Unix and Windows side by side

You can also download the CD or DVD and install on your everyday desktop computer. Most of the time (actually virtually all the time, but we're making no promises here), you can install Unix onto your desktop right alongside your Windows environment without breaking anything. You'll get it installed, reboot your system, and choose Unix (Linux, OpenSolaris, whatever) or Windows when you boot up. This option isn't bad, but it does require you to stop what you're doing in Windows or Unix to change to the other. If your desktop computer is relatively old, this might be better than the following options, though.

If you have a pretty beefy desktop computer (relatively new with ample memory and disk space), you could try using Sun Microsystems' VirtualBox or VMWare, VirtualPC, or other virtualization environments, which give you *computer emulation* (think "picture in picture" for your computer, but with one operating system within the other operating system).

continues on next page

Many of the examples and screenshots for this book were taken from Unix systems running under VirtualBox on one of our desktop systems.

Cygwin provides you with a Unix environment that's actually part of your Windows system. It takes a bit of getting used to, but Cygwin is stable and reliable. The hardest part about using Cygwin is that it can be confusing to know whether you're dealing with Unix or Windows at any given moment.

Different Unix flavors

So, given all of those options for getting access to Unix, the choice of which kind of Unix (which Unix *flavor*) must be clear and straightforward—right? Of course not.

If you're just getting started with Unix, we recommend having you choose the flavor that your most techie friends or the folks at work use. This will give you potential built-in tech support options.

If you're starting purely from scratch, look into the most popular and highly rated Linux distributions. (Currently, the Web site `www.distrowatch.com` provides a great set of recommendations, but as you know, Web sites change, so you might want to also do some Web searching for recommended Linux distributions.)

A newly popular (or popular again) option is OpenSolaris, from Sun Microsystems. For a while Solaris was a bit tricky (well, a lot tricky) to get installed and functional on a regular desktop system; however, it's now as easy as the easier Linux systems, and it offers a tremendous amount of power and flexibility, in addition to some cutting-edge technologies.

That said, any option you choose will be pretty similar for the purposes of this book. Differences among the options primarily show up in more advanced applications.

✔ Tip

- If you're using Mac OS X or later, you're already using Unix—you just need to bring up a terminal window to be able to follow right along with the book.

About Connecting

Once upon a time, when dinosaurs roamed the earth, Unix users connected to their systems using `telnet`. With `telnet`, your password and everything else you do is sent straight across the wire and can be easily read by anyone on the same part of the network. Yikes is right! That's why, more and more, ISPs and system administrators require something called `ssh` (Secure SHell) to connect to their systems. With `ssh`, everything is encrypted, precisely the way your Web connection is encrypted when you use an e-commerce site and see that the little padlock in your Web browser is closed.

Yes, we know, you don't have any secrets, but if a hacker logs into your ISP's system as you, that same hacker has won 50 percent of the battle for taking over that system for any number of illegal activities. And, if your neighbor's 19-year-old son *sniffs* (that's the technical term) your user identification (often called the userid or user ID) and password over your cable modem connection (and that's entirely possible), he can probably guess that your eBay password, broker password, or whatever are the same or at least similar.

Throughout this book, we'll show examples using an `ssh` connection. If, for whatever reason, your system administrators don't require `ssh`, we recommend using it anyway; there is absolutely no reason not to, because there are no disadvantages to `ssh` compared to `telnet`. If your systems don't support `ssh`, you can use the `telnet` or `rlogin/rsh` program as alternatives.

Connecting to the Unix System

Your first step in using Unix is to connect to the Unix system. Exactly how you connect will vary depending on what kind of Internet connection you use, but the following steps should get you started.

To connect to the Unix system:

1. Connect to the Internet, if necessary.

   If you have to start your Internet connection manually, launch it now. If you use a full-time Internet connection at home, work, or school, or if you're using your Mac or Linux system at home, just ignore this step.

2. If you're connecting to a remote system, start your `ssh` program and connect to the Unix system.

   Using `ssh` you can connect to a remote computer (such as your ISP's computer) and work as if the remote computer were sitting on your desk. Essentially, `ssh` brings a remote computer's capabilities to your fingertips, regardless of where you're physically located. (See the "About Connecting" sidebar for more information about connection technologies.)

   Exactly how you connect depends on the particular program you're using. For Windows users, we recommend PuTTY, which is a free `ssh` client available at `www.chiark.greenend.org.uk/~sgtatham/putty/`. For Macintosh users (pre–OS X), we recommend the predictably named `MacSSH`, also free, available at `http://sourceforge.net/projects/macssh`.

continues on next page

And, of course, after you're logged into your Unix-like system, you can use the Unix `ssh` command to access other computers. Each program works a bit differently, and you'll have to refer to the specific documentation for details about using them.

In this example, we're connecting to a Unix system using PuTTY. **Figure 1.1** shows the Configuration dialog box, in which we've specified Host Name (frazz.raycomm.com), Port (22), and Protocol (`ssh`).

If you're looking for a quick start, just fill in the fields shown in **Figure 1.2** and click Open.

3. Alternatively, if you're on a Mac or Linux or Unix system already, just open a terminal window and you'll be all set—and you won't even have to log in.

4. Check out the Categories (or the Preferences dialog box in many other programs) and become familiar with your options. You will not need to change anything initially, but you might later want to customize colors or other settings. Generally, though, PuTTY provides usable settings.

5. Marvel at the `login:` prompt, which is what you should see if you've connected properly (**Figure 1.3**) and move along to the next section. (PuTTY displays "login as :", while most other programs will just show you "login:". Don't worry about this difference; it's just this program's idiosyncrasy.)

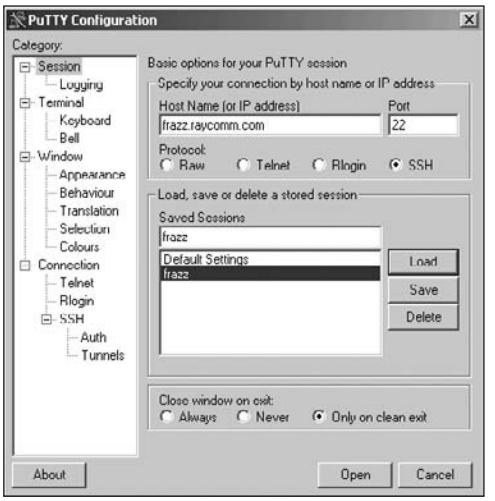

Figure 1.1 Here we're connecting to `frazz.raycomm.com` using PuTTY. Other `ssh` programs might look slightly different, but this shows the general idea.

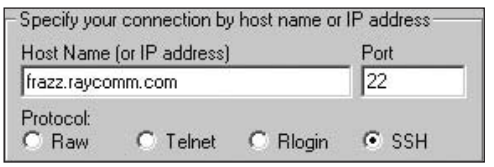

Figure 1.2 For a quick start, fill in these fields, and then click Open.

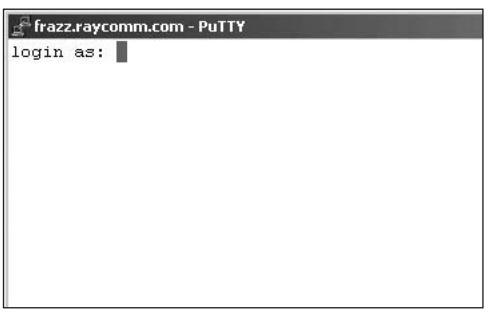

Figure 1.3 PuTTY shows a login as: prompt from `frazz.raycomm.com`.

Before You Begin

Before you begin, have your connection information, such as your login name and password, handy.

Contact your system administrator if you don't yet have these. Throughout this book, we'll use "system administrator" to refer to your help desk, ISP technical support line, or anyone else you can call on who runs your Unix system and can help you. Sometimes that geeky daughter, brother, or otherwise Unixy-person can help you out with Unix, too; however, in many cases you'll find that you need to troubleshoot a problem with the person who can manage *your* account information.

✔ Tips

- If you modify the connection settings, you may need to disconnect from the session, then reconnect again for the new settings to take effect. See your documentation for specifics about disconnecting from your session.
- In addition to viewing the buffer to see commands you've used, as mentioned in the "The SSH Preferences Dialog Box" sidebar (later in this chapter), you can also use a command to let you review commands that you've issued. For more information, see the appropriate "Viewing Session History" section in Chapter 3.

Write Down Details About Your Specific login Procedure

As you go through your login procedure, take a minute to write down some details for future reference.

Your userid or login name (but not your password):

_____

The name of the program you use (or the icon you click) to connect to your Unix system and the process you use to get connected:

_____

_____

_____

_____

The name of your Unix system (such as `frazz.example.com` or `example.com`):

_____

The IP (Internet Protocol) address of your Unix system (such as `198.168.11.36` or `10.10.22.2`):

_____

Logging In

After you've connected to the Unix system, your next step is to *log in*, or identify yourself to the Unix system. Logging in serves a few purposes, including giving you access to your e-mail, files, and configurations. It also keeps you from inadvertently accessing someone else's files and settings, and it keeps you from making changes to the system itself.

To log in:

1. Have your userid (user identification) and password ready.

   Contact your system administrator if you don't have these yet.

2. Type your userid at the login prompt, then press Enter.

   Your userid is case sensitive, so be sure you type it exactly as your system administrator instructed.

3. Type your password at the password prompt, then press Enter.

   Yup. Your password is case sensitive, too.

4. Read the information and messages that come up on the screen.

   The information that pops up—the message of the day—might be just funny or lighthearted, as in **Figure 1.4**, or it might contain information about system policies, warnings about scheduled downtime, or useful tips, as shown in **Figure 1.5**. It may also contain both, or possibly neither, if your system administrators have nothing to say to you.

After you've logged in, you'll see a *shell prompt*, which is where you type in commands. Also, note that you'll be located in your *home directory*, which is where your personal files and settings are stored. Your "location" in the Unix system is a slightly unwieldy concept that we'll help you understand throughout this chapter.

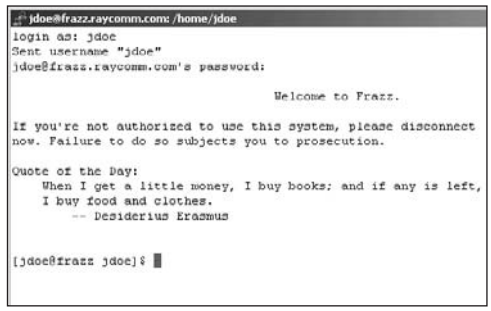

Figure 1.4 Our Unix system (`frazz.raycomm.com`) greets us with a quote of the day, called a "fortune."

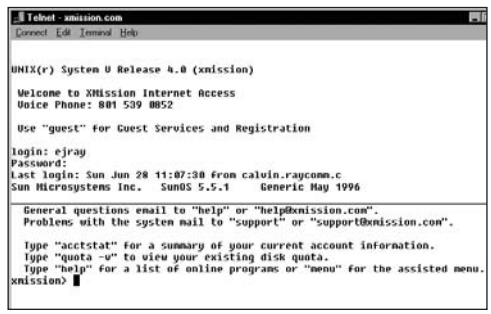

Figure 1.5 Some systems might greet you with system information or helpful tips.

✔ Tips

- If you get an error message after attempting to log in, just try again. You likely just mistyped your userid or password. Whoops!
- When you log in, you might see a message about failed login attempts. If you unsuccessfully tried to log in, then don't worry about it; the message just confirms that you attempted to log in but failed. If, however, all of your login attempts (with you sitting at the keyboard) have been successful or if the number of failed login attempts seems high—say, five or more—then you might also mention the message to your system administrator, who can check security and login attempts. This could be a warning that someone unauthorized is trying to log in as you.

```
$ passwd
Changing password for ejr
(current) Unix password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated
→ successfully
$
```

Code Listing 1.1 Change your password regularly using the `passwd` command.

The SSH Preferences Dialog Box

In the SSH Preferences dialog box, you can fix some of the idiosyncrasies that are caused by how your `ssh` program talks to the Unix system. You can't identify these idiosyncrasies until you actually start using your Unix system, but you should remember that you can fix most problems here. For example:

- If your ←Backspace and Delete keys don't work, look for an option in your `ssh` or `telnet` program that defines these keyboard functions.
- If you start typing and nothing shows up onscreen, set local echo to on.
- If you start typing and everything shows up twice, set local echo to off.
- If you want to be able to scroll up onscreen to see what's happened during your Unix session, change the buffer size to a larger number.

Exactly which options you'll have will vary from program to program, but these are ones that are commonly available. Click OK when you're done playing with the settings.

Changing Your Password with *passwd*

Virtually all Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and *crackers* (malicious hackers). **Code Listing 1.1** shows how you change your password.

Throughout your Unix adventure, you'll likely change your password often:

- You'll probably want to change the password provided by your system administrator after you log in for the first time. Hint, hint.
- You'll probably change your password at regular intervals. Many Unix systems require that you change your password every so often—every 30 or 60 days is common.
- You might also change your password voluntarily if you think that someone might have learned it or if you tell anyone your password (although you really shouldn't do that anyway).

To change your password:

1. `passwd`

   To start, type `passwd`.

2. `youroldpassword`

   Enter your old password—the one you're currently using. (Of course, type in your old password, not the sample one we've used here!) Note that the password doesn't show up onscreen when you type it, in case someone is lurking over your shoulder, watching you type, and asking, "Whatcha doing?"

continues on next page

3. `yournewpassword`

   Type your new password. Check out the "Lowdown on Passwords" sidebar for specifics about choosing a password.

4. `yournewpassword`

   Here, you're verifying the password by typing it again.

   The system will report that your password was successfully changed (specific terminology depends on the system) after the changes take effect. This is also shown in Code Listing 1.1.

✔ Tips

- Double-check your new password before you log out of the system by typing `su - yourid` at the prompt. Of course, substitute your real username (or login name) for `yourid` here. This command (switch user) lets you log in again without having to log out, so if you made a mistake when changing your password and now get a failed login message, you can find out before you actually disconnect from the system. If you have problems, contact your system administrator before you log out so you can get the problem resolved.
- In some environments, you will use `yppasswd`, not `passwd`, to change your password, or even use a Web page or other means. When in doubt, defer to what your system administrator told you to do. ("The Rays said to use this other command" is likely to get all of us in trouble.)

The Lowdown on Passwords

In addition to following any password guidelines your system administrator mandates, you should choose a password that is

- At least six characters long
- Easy for you to remember
- Not a word or name in any dictionary in any language
- A combination of capital and lowercase letters, numbers, and symbols
- Not similar to your username
- Not identical or similar to one you've used recently
- Not your telephone number, birth date, kid's birth date, anniversary (even if you can remember it), mother's maiden name, or anything else that someone might associate with you

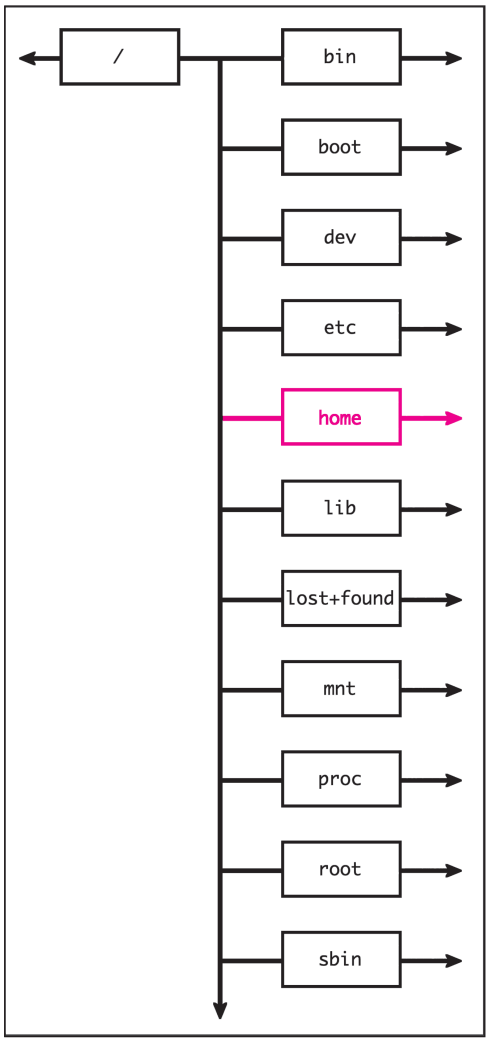

Figure 1.6 All files and directories are nested within the root directory, which serves to contain everything in the system.

```
[jdoe@frazz jdoe]$ ls
limerick  mail/  Project/  public_html/
→ testfile  testlink@  tmp/
[jdoe@frazz jdoe]$
```

Code Listing 1.2 Use `ls` by itself to list the files and subdirectories of the directory you're in.

Listing Directories and Files with `ls`

Your Unix system is made up of directories and files that store a variety of information, including setup information, configuration settings, programs, and options, as well as other files and directories. You might think of your Unix system as a tree (tree roots, actually), with subdirectories stemming from higher-level directories. As shown in **Figure 1.6**, all of these files and directories reside within the root directory, which contains everything in the system.

Using the `ls` command, you can find out exactly what's in your Unix system and thereby find out what's available to you. You can list the files and directories of a directory that you're currently in or a directory that you specify.

To list the files and directories of the directory you're in:

- `ls`

  At the shell prompt, type `ls` to list the files and directories in the current directory, which in this case is our home directory (**Code Listing 1.2**).

To list the files and directories of a specified directory:

- `ls /bin`

  Here, you type the `ls` command plus the name of a directory. As shown in **Code Listing 1.3**, this command lists the files and directories in the `/bin` directory, in which you'll find system commands and programs.

✔ Tips

- You can list the files and directories of the *root directory* at any time and in any place by typing `ls /`. The root directory is the highest-level directory in a Unix system; all other directories are below the root directory.
- Can't remember that pesky filename? Just use `ls` to help jog your memory. Or, refer to "Finding Forgotten Files with `find`" in Chapter 2, which can also help you remember filenames.
- Many other `ls` options are available to control the amount of information about your files that you see and the format in which they appear onscreen. See Chapter 2's "Listing Directories and Files with `ls` (More Goodies)" section for details.


```
[jdoe@frazz jdoe]$ ls /bin
arch*              domainname@        ipcalc*            open*         tar*
awk@               echo*              ip6c               eys*          red@          unlink*
chmod*             fbresolution*      login*             rm*           usleep*
chown*             fgrep@             ls*                rmdir*        vi@
consolechars*      find*              lsb_release*       rpm*          view@
cp*                gawk*              mail*              rvi@          vim@
cpio*              gawk-3.1.1@        mkdir*             rview@
date*              gtar@              more*              sleep*        zcat*
dd*                gunzip*            mount*             sort*         zsh*
df*                gzip*              mv*                stat*
dmesg*             hostname*          netstat*           stty*
dnsdomainname@     id*                nice*              su*
doexec*            igawk*             nisdomainname@     sync*
[jdoe@frazz jdoe]$
```

Code Listing 1.3 Use `ls` with the name of a directory to list the contents of that directory (`/bin`, in this case).

```
[jdoe@frazz jdoe]$ cd /
[jdoe@frazz /]$ cd
[jdoe@frazz jdoe]$ cd /home/jdoe/Project/
[jdoe@frazz Project]$ cd /etc
[jdoe@frazz etc]$ cd /home/jdoe/
[jdoe@frazz etc]$ cd /home/jdoe/mail/
[jdoe@frazz mail]$ cd ..
[jdoe@frazz jdoe]$
```

Code Listing 1.4 Using cd, you can change directories and move around in the system. Note that the prompt in this code listing shows the name of the current directory, which can be handy.

Changing Directories with cd

To explore Unix and its capabilities, you'll need to move around among the directories. You do so using the `cd` command, which takes you from the directory you're currently in to one that you specify. **Code Listing 1.4** illustrates how you use `cd` to change directories.

To change directories:

1. `cd Projects`

   To move to a specific directory, type `cd` plus the name of the directory. In this example, we move down in the directory tree to a subdirectory called `Projects`. (See the "Moving Up and Down" sidebar for an explanation of what "up" and "down" mean in Unix terms.)

2. `cd ..`

   Type `cd ..` to move up one level in the directory tree.

3. `cd /etc`

   Here, `/etc` tells the system to look for the `etc` directory located at the system root.

Moving Up and Down

Throughout this book, we'll talk about moving "up" and "down" through the Unix file system. Moving "up" means moving into the directory that contains the current directory—that is, closer to the root directory. Moving "down" means moving into subdirectories that are contained by the current directory—that is, further from the root directory.

✔ Tips

- If you don't remember the name of the directory you want to change to, you can use `ls` to list the directories and files in your current directory, then use `cd` as shown earlier. See the previous section, "Listing Directories and Files with `ls`," for more information.
- You can return to your home directory from anywhere in the Unix system by entering `cd` without specifying a directory.
- You can often use a tilde (~) as a handy shortcut to your home directory. For example, if you want to change to the `Urgent` directory within the `Projects` directory in your home directory, you could use something like `cd /home/users/y/yourid/Projects/Urgent` or just use the shortcut `cd ~/Projects/Urgent`.
- Keep in mind that your home directory isn't the same as the system root directory. You might think of your home directory as "the very small section of the Unix system that I can call my own." Every person using the Unix system has his or her own little personal section.

  The current directory is always indicated with a `.`, while the next higher directory (the one that contains the current directory) is indicated with `..` (two dots). That is why you use `cd ..` to move up a directory. In Chapter 10, you will see a specific use for `.` to specify the current directory when running scripts or programs.
- Visit Chapter 2 for much more about directories and files.

```
[jdoe@frazz jdoe]$ pwd
/home/jdoe
[jdoe@frazz jdoe]$ ls ; pwd
codelisting1.2  codelisting1.4  mail/
→ public_html/  testlink@
codelisting1.3  limerick  Project/  testfile
→ tmp/
/home/jdoe
[jdoe@frazz jdoe]$ cd
[jdoe@frazz jdoe]$ cd /
[jdoe@frazz /]$ pwd
/
[jdoe@frazz /]$
```

Code Listing 1.5 pwd displays the name of the current directory, which is particularly handy if you've been exploring the system. By combining commands, you can request the directory's name and contents at one time.

Finding Yourself with pwd

As you begin using Unix and start moving around in directories and files, you're likely to get a bit lost—that is, forget which directory or subdirectory you're in. You can use the `pwd` command to get a reminder of where you are, as shown in **Code Listing 1.5**.

You can request just the directory name, or you can get fancy and request the directory's name and its contents, courtesy of `ls`.

To find out the name of the current directory:

- `pwd`

  This command displays the path and name of the directory you are currently in. The path names each of the directories "above" the current directory, giving you the full picture of where you are in relationship to the system root.

To find out the name of the current directory and its contents:

- `ls ; pwd`

  By combining the `ls` and `pwd` commands, you can request the directory's contents and name, as shown in Code Listing 1.5.

✔ Tips

- Type `pwd` immediately after you log in. You'll see where your home directory is in the overall system (aka the full path name for your home directory).
- On some Unix systems, you won't need to use `pwd` to find out where you are. Some systems display the current directory at the shell prompt by default—something like `/home/ejr>`. If you'd like to add or get rid of this, or if you want more information about shells and customizing your shell, see Chapter 8.

Piping Input and Output

In general, you can think of each Unix command (`ls`, `cd`, and so on) as an individual program that Unix executes. For example, if you type `cat /etc/motd` at the prompt, Unix will display the contents of `motd` in the `/etc` directory. Each program requires input (in this example, `cat`, the program, takes the contents of `/etc/motd` as input) and produces output (i.e., the displayed results).

Frequently, you'll want to run programs in sequence. For example, you could tell Unix to read your resume and then spell-check it. In doing this, you connect two commands together and have them run in sequence. This process, in which you connect the output of one program to the input of another, is called piping. Depending on what you want to do, you can pipe together as many commands as you want—with the output of each command acting as the input of the next.

As **Figure 1.7** shows, you pipe commands together using the pipe symbol, which is the `|` character. In the following example, we'll pipe the output of the `ls` command (which lists the contents of a directory) to the `more` command (which lets you read results one screen at a time). For details about `more`, see "Viewing File Contents with `more`," later in this chapter.

Figure 1.7 To execute multiple commands in sequence, pipe them together using the pipe symbol (`|`).

To pipe commands:

- `ls | more`

  Here, all you do is include a pipe symbol between the two commands, with or without a space on both sides of the pipe. This code produces a list of the files in the current directory, then pipes the results to `more`, which then lists the results one screen at a time (see Figure 1.7).

✔ Tips

- If you want to pipe more than two commands, you can. Just keep adding the commands (with a pipe symbol in between each, `like | this`) in the order you want them executed.
- Remember that the output of each command is piped to the next command. So a piped command, such as `ls | spell | sort`, could list files within a directory, then spell-check the list, then sort the misspelled words and display them onscreen. The filenames that are found in the system dictionary would not appear.
- Venture to Chapter 15 to find out more about running a spell-checker and Chapter 6 to find out more about sorting.

Redirecting Output

Suppose you've developed your resume and spell-checked it. As you learned in the previous section, the results you see onscreen will be the output of the last command—in this case, a list of misspelled words. A lot of times, you'll want to redirect the final output to another location, such as to a file or a printer (if a printer is an option for you), rather than view it onscreen. You can do this using *redirection*, which sends the final output to somewhere other than your screen.

As shown in **Code Listing 1.6**, you will often redirect output results to a file. Notice the greater-than symbol (>), which indicates that the output of the program is to be redirected to the location (or filename) you specify after the symbol.

In the following examples, we'll show you how to redirect output to a new file and how to redirect output to append it to an existing file.


```
[jdoe@frazz jdoe]$ ls /usr/local/bin > local.programs.txt
[jdoe@frazz jdoe]$ ls local*
localize  localono  local.programs.txt  localyokel
[jdoe@frazz jdoe]$ ls /usr/bin >> other.programs.txt
[jdoe@frazz jdoe]$
```

Code Listing 1.6 In this case, the output of `ls` gets redirected to `local.programs.txt`, as indicated by the greater-than (>) symbol. The asterisk wildcard (*) acts as a placeholder for letters or numbers. Finally, the listing of `/usr/bin` gets appended to the `other.programs.txt` file.

To redirect output to a new file:

1. `ls /usr/local/bin > local.programs.txt`

   In this case, we start with the `ls` command and a specific directory, add a greater-than symbol (>), and then specify a filename. This will redirect the output of `ls` to a file named `local.programs.txt`.

   Be careful with this command! If the file already exists, it could be replaced with the output of the `ls` program here.

2. `ls local*`

   Here, we're just checking to see that the new `local.programs.txt` file has successfully been created. The asterisk wildcard (`*`) specifies that we want a list of all files that begin with the word `local`, such as `localize`, `localyokel`, or `localono` (see Code Listing 1.6). See the next section, "Using Wildcards," for handy wildcard information.

To append output to an existing file:

- `ls /usr/bin >> all.programs.txt`

  Appending output to an existing file is similar to redirecting it to a new file; however, instead of creating a new file to hold the output (or replacing the contents of an existing file), you add content to the end of an existing file. Notice that you use two greater-than symbols here, rather than one.

✔ Tip

- You can pipe and redirect at the same time. For example, you might list a directory, pipe it to `wc` to count the entries, then append the results to a `directoryinfo` file, like this: `ls | wc -l >> directoryinfo`. You can learn more about counting files and their contents with `wc` in Chapter 6.

```
[jdoe@frazz Project]$ ls
keep keeper.jpg   kept   kidder.txt
→ kiddo    kidnews   kidneypie
→ kids     kidupdate
[jdoe@frazz Project]$ ls ki*
kidder.txt    kiddo    kidnews   kidneypie
→ kids  kidupdate
[jdoe@frazz Project]$ ls kid*
kidder.txt    kiddo    kidnews   kidneypie
→ kids  kidupdate
[jdoe@frazz Project]$ ls k???
keep kept   kids
[jdoe@frazz Project]$ ls *date
kidupdate
[jdoe@frazz Project]$ ls *up*
kidupdate
[jdoe@frazz Project]$ ls k?d*
kidder.txt    kiddo    kidnews   kidneypie
→ kids     kidupdate
[jdoe@frazz Project]$
```

Code Listing 1.7 You use wildcards (? or *) to act as placeholders for missing characters.

Using Wildcards

You might think of wildcards as placeholders for omitted letters or numbers. For example, if you're looking for a file but aren't sure whether you named it `kidnews` or `kidupdate`, you can include a wildcard to stand for the part you're uncertain of. That is, you could list the files of a directory with `ls kid*` (**Code Listing 1.7**), which would list all files starting with the characters *kid*. In the resulting list, you'd find a file named `kid` if there were one, as well as files that begin with `kid` but have varying endings, such as `kidnews` (aha, the lost file!), `kiddo`, or `kidneypie`.

You can use wildcards for just about any purpose in Unix, although listing files and directories will likely be the most common use. Just follow these guidelines:

- You use ? as a placeholder for one character or number.
- You use * as a placeholder for zero or more characters or numbers. Zero characters, in case you're curious, specifies that the search results include all variants of `kid`, including the word itself with no suffix.
- You can include a wildcard at any place in a name: at the beginning (`*kid`), somewhere in the middle (`k*d`), at the end (`ki*`), or even in multiple places (`*kid*`).

Viewing File Contents with `more`

As you become more familiar with Unix, you'll want to start exploring the contents of files, including some program files and scripts as well as files you eventually create. One of the easiest ways to view file contents is to use the `more` command, which tells Unix to display files onscreen, a page at a time. As shown in **Figure 1.8**, long files are displayed with "More" at the bottom of each screen so that you can move through the file one screen at a time using the spacebar.

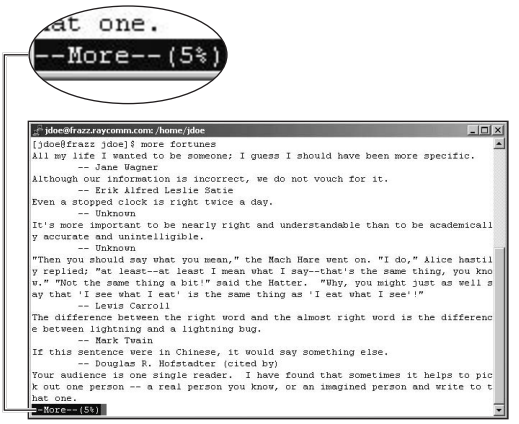

Figure 1.8 The `more` command lets you move through a file one screen at a time, providing a "More" indicator at the bottom of each screen.

To view a file with `more`:

1. `more fortunes`

   At the prompt, type `more` plus the name of the file you want to view. You'll see the contents of the file you requested, starting at the top (Figure 1.8).

2. Spacebar

   Press Spacebar to see the next screen of information. As you move through the file, you can press B to move back through previous screens.

3. Q

   When you're done, press Q to go back to the shell prompt.

✔ Tips

- If you want to view just an additional line (rather than an entire screen) when using `more`, press Enter instead of the Spacebar.

  You can also use `less` to view files. `less` is similar to `more`, but it's more powerful and flexible. How can `less` be more and `more` be less? As you'll see in Appendix C: "Commands and Flags," the `more` command has 10 options or so; the `less` command has about 40.

- You can also view files using the `cat` command. See the next section for the full scoop.

```
[jdoe@frazz jdoe]$ cat newest.programs.txt
...
xpmtoppm*
xpp*
xpstat*
xrfbviewer*
xscreensaver.kss*
xvminitoppm*
xwdtopnm*
xxd*
yaf-cdda*
yaf-mpgplay*
yaf-splay*
yaf-tplay*
yaf-vorbis*
yaf-yuv*
ybmtopbm*
yelp*
yes*
ypcat*
ypchfn*
ypchsh*
ypmatch*
yppasswd*
ypwhich*
yuvsplittoppm*
yuvtoppm*
z42_cmyk*
z42tool*
zcmp*
zdiff*
zeisstopnm*
zforce*
zgrep*
zipgrep*
zipinfo*
zless*
zmore*
znew*
[jdoe@frazz jdoe]$ cat newer.programs.txt
→ newest.programs.txt > all.programs
[jdoe@frazz jdoe]$
```

Code Listing 1.8 With `cat`, long files whirl by, and all you'll see is the bottom of the file. You can also redirect `cat` output to a file, as shown at the end of the listing.

Displaying File Contents with `cat`

Instead of using `more` to display files, you can use `cat` (as in "con*cat*enate"), which displays files but does not pause so you can read the information. Instead, it displays the file or files—which whiz by onscreen—and leaves you looking at the last several lines of the file (**Code Listing 1.8**).

The `cat` command also lets you redirect one or more files, offering a function that some versions of `more` do not.

To display file contents with `cat`:

- `cat newest.programs.txt`

  To begin, type `cat` plus the filename (probably not `newest.programs` unless you're naming your files just like we are).

  The file contents will appear onscreen; however, if the file is longer than a single screen, the contents will whirl by, and all you'll see is the bottom lines of the file—the 24 or so that fit on a single screen.

  or

- `cat newer.programs.txt newest.`
  `→ programs.txt`

  You can also specify multiple files for `cat`, with each file displayed in the order specified. In this example the contents of `newer.programs` will zip by, then the contents of `newest.programs` will zip by.

continues on next page

or

- `cat newer.programs.txt newest.`
  `→programs.txt > all.programs`

  In this example, we've added a redirection symbol (>) plus a new filename. This tells Unix to print out both files; however, instead of displaying the files onscreen, it redirects them to the file called `all.programs`. Aha! Here's where `cat` does something better than `more`. See "Redirecting Output," earlier, for more information about redirecting commands.

✔ Tips

- If you inadvertently use `cat` with a *binary file* (a nontext file), you might end up with a whole screen of garbage. On some systems, you might try `stty sane` or reset to fix it—more on this in "Fixing Terminal Settings with `stty`" in Chapter 3. You could also just close your terminal window and log in again to fix it.
- The `tac` command is just like `cat`, but backward. Try it! Oddly handy, eh?
- You can also view file contents using the `more` command. See the previous section for details.

Table 1.1

Common Unix Directories and Their Contents

Directory	Contents
/bin	Essential programs and commands for use by all users
/etc	System configuration files and global settings
/home	Home directories for users
/sbin	Programs and commands needed for system boot
/tmp	Temporary files
/usr/bin	Commands and programs that are less central to basic Unix system functionality than those in /bin but were installed with the system or that came as part of the distribution
/usr/local	Most files and data that were developed or customized on the system
/usr/local/bin	Locally developed or installed programs
/usr/local/man	Manual (help) pages for local programs
/usr/share/man	Manual (help) pages
/var	Changeable data, including system logs, temporary data from programs, and user mail storage

Exploring the System

With these few key skills in hand, you're ready to start exploring your Unix system. In doing so, you can quickly get an idea of what's available and gain some useful experience in entering commands.

Think of your Unix system as a thoroughly kid-proofed house: You can look around and touch some stuff, but you can't do anything to hurt yourself or the system. So, don't worry! You can't hurt anything by looking around, and even if you tried to break something, most Unix systems are configured well enough that you couldn't.

Table 1.1 shows some of the directories you're likely to find most interesting or useful (Appendix B of this book provides a more comprehensive list of directories). You can use the following steps to get started exploring.

To explore locally installed programs:

1. `cd /usr/bin`

   Change to `/usr/bin`, which is where most installed programs are.

2. `ls | more`

   List the files (which will be programs, in this example) and pipe the output to `more` so you can read the names one screen at a time.

3. `ssh`

   Type the name of any program you want to run; `ssh`, in this case, allows you to connect to another system and use it just as you're using your Unix system now.

✔ Tip

- You can type `man` followed by a command name to learn more about Unix programs. See the next section for information about Unix help.

Getting Help with man

Occasionally, you may need a bit of help remembering what a particular command does. Using man (which is short for "manual"), you can look up information about commands and get pointers for using them efficiently. **Figure 1.9** shows a Unix help page (also called a man page, for obvious reasons) for passwords. In the following steps, we'll show you how to look up specific Unix commands and find related topics.

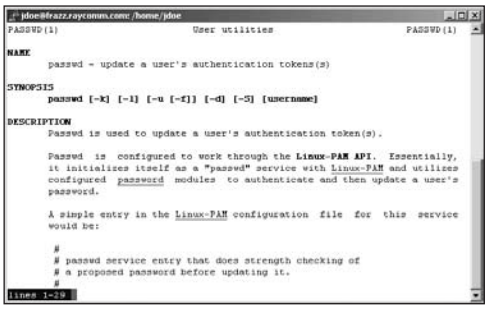

Figure 1.9 Using man passwd, you can access the standard man file about the passwd program.

To access a man page:

- man passwd

  At the prompt, type man plus the name of the command you want help with (in this case, passwd). You'll get the man page for that command. Use the Spacebar and the B key (for Back) to navigate through the file, just as you do with more.

To find a specific man page:

1. man -k passwd

   Type man -k plus the name of the command or the topic you want help with (in this case, passwd). As **Code Listing 1.9** shows, you'll see a list of possible man pages: command names, man page names, and a description. Note the man page name (and number if more than one page with the same name exists) so you can reference it in the next step.

```
$ man -k passwd
chpasswd (8)  - update password file in batch
gpasswd (1)   - administer the /etc/group file
mkpasswd (1)  - generate new password, optionally apply it to a user
mkpasswd (8)  - Update passwd and group database files
passwd (1)    - update a user's authentication tokens(s)
passwd (5)    - password file
userpasswd (1)    - A graphical tool to allow users to change their passwords
```

Code Listing 1.9 man -k passwd gives you these results, showing specific password-related man pages.

2. `man 1 passwd`

   Here, you type `man`, the `man` page you want to view (indicated by 1 in this case to specify section 1—this is necessary because more than one `man` page with the name `passwd` was listed in the last step), and the command name (`passwd`). Figure 1.9 shows the resulting `man` page.

✔ Tips

- You can make a copy of a `man` page so you can edit it or comment on it, adding additional notes for your information or deleting irrelevant (to you) stuff. Just type `man commandname | col -b -x > somefilename`. For example, use `man passwd | col -b -x > ~/my.password.command.notes` to make a copy of the `passwd man` page, sans formatting, in your home directory, under the name `my.password.command.notes`. Then you'll use an editor (from Chapter 4) to edit, add to, and tweak the important points. (The `col -b -x` command fixes some formatting oddities; without it, all of the underlined words might show up as `_u_n_d_e_r_l_i_n_e`, depending on the system.)
- You can use `apropos` instead of the `man -k` flag. For example, you might use this: `apropos passwd`.
- Some Unix systems might require a `-s` before the section number, as in `man -s 1 passwd`.

Logging Out

When you finish your session, you need to log out of the system to ensure that nobody else accesses your files while masquerading as you.

To log out:

- `logout`

  That's it! Just type `logout`, and the system will clean up everything and break the connection, and the `ssh` program might very well just vanish completely.

✔ Tip

- On some Unix systems, you can type `exit` or `quit` instead of `logout`, or press Ctrl D on your keyboard.

Using Directories and Files

2

As you learned in Chapter 1, directories and files are the heart of Unix; they contain things like setup information, configuration settings, programs, and options, as well as anything that you create. You access directories and files every time you type in a Unix command, and for this reason, you need to become familiar with the various things you can do with them.

Again in this chapter, the skills and commands we'll cover apply to any Unix flavor. What you see onscreen (particularly system prompts and responses) may differ slightly from what's illustrated in this book. The general ideas and specific commands, however, will be the same on all Unix systems.

Chapter Contents

Creating Directories with `mkdir`

You might think of directories as being drawers in a file cabinet; each drawer contains a bunch of files that are somehow related. For example, you might have a couple of file drawers for your unread magazines, one for your to-do lists, and maybe a drawer for your work projects.

Similarly, directories in your Unix system act as containers for other directories and files; each subdirectory contains yet more related directories or files, and so on. You'll probably create a new directory each time you start a project or have related files you want to store at a single location. You create new directories using the `mkdir` command, as shown in **Code Listing 2.1**.


```
$ ls
Projects  all.programs.txt    local.programs.txt         schedule
Xrootenv.0    files    newer.programs  short.fortunes
all.programs  fortunes     newest.programs     temp
$ mkdir Newdirectory
$ ls -l
total 159
drwxrwxr-x       2 ejr       users       1024 Jun 29 11:40 Newdirectory
drwxrwxr-x       2 ejr       users       1024 Jun 28 12:48 Projects
-rw-rw-r-        1 ejr       users       7976 Jun 28 14:15 all.programs
-rw-rw-r-        1 ejr       users       7479 Jun 28 14:05 all.programs.txt
-rw-rw-r-        1 ejr       users       858 Jun 28 12:45 files
-rw-rw-r-        1 ejr       ejr         128886 Jun 27 09:05 fortunes
-rw-rw-r-        1 ejr       users       0 Jun 28 14:05 local.programs.txt
-rw-rw-r-        1 ejr       users       497 Jun 28 14:13 newer.programs
-rw-rw-r-        1 ejr       users       7479 Jun 28 14:13 newest.programs
lrwxrwxrwx       1 ejr       users       27 Jun 26 11:03 schedule -> /home/deb/Pre
-rw-rw-r-        1 ejr       ejr         1475 Jun 27 09:31 short.fortunes
drwxrwxr-x       2 ejr       users       1024 Jun 26 06:39 temp
$
```

Code Listing 2.1 Typing `mkdir` plus a directory name creates a new directory. Listing the files, in long format, shows the new directory. The "d" at the beginning of the line shows that it's a directory.

Naming Directories (and Files)

As you start creating directories (and files), keep in mind the following guidelines:

- Directories and files must have unique names. For example, you cannot name a directory `Golf` and a file `Golf`. You can, however, have a directory called `Golf` and a file called `golf`. The difference in capitalization makes each name unique. By the way, directories are often named with an initial cap, and filenames are often all lowercase.
- Directory and filenames can, but should not include the following characters: angle brackets (`< >`), braces (`{ }`), brackets (`[ ]`), parentheses (`( )`), double quotes (`" "`), single quotes (`' '`), asterisks (`*`), question marks (`?`), pipe symbols (`|`), slashes (`/ \`), carets (`^`), exclamation points (`!`), pound signs (`#`), dollar signs (`$`), ampersands (`&`), and tildes (`~`).

  Different shells handle special characters differently, and some will have no problems at all with these characters. Generally, though, special characters are more trouble than they're worth.
- Generally, avoid names that include spaces. Some programs don't deal with them correctly, so to use spaces you have to use odd workarounds. Instead, stick to periods (`.`) and underscores (`_`) to separate words, characters, or numbers.
- Use names that describe the directory's or file's contents so you easily remember them.

To create a directory:

1. `ls`

   Start by listing existing directories to make sure that the planned name doesn't conflict with an existing directory or filename.

2. `mkdir Newdirectory`

   Type the `mkdir` command to make a new directory; in this case, it's called `Newdirectory`. Refer to the sidebar "Naming Directories (and Files)" for guidelines.

3. `ls –l`

   Now you can use `ls -l` (the `-l` flag specifies a long format) to look at the listing for your new directory (Code Listing 2.1). The `d` at the far left of the listing for `Newdirectory` indicates that it's a directory and not a file. Of course, after you trust Unix to do as you say, you can skip this verification step.

✔ Tips

- If you attempt to create a directory with a file or directory name that already exists, Unix will not overwrite the existing directory. Instead, you'll be told that a file by that name already exists. Try again with a different name.
- You can create several directories and subdirectories at once with the `–p` flag. For example, if you want to create a new subdirectory called `Projects` with a subdirectory called `Cooking` within that and a subdirectory called `Desserts` within that, you can use `mkdir –p Projects/Cooking/Desserts` and get it all done at once. Without the `–p` flag, you have to create `Projects`, `Cooking`, then `Desserts` in order, which is a longer recipe to make the same tree structure.

Creating Files with touch

Another skill you'll use frequently is creating files. You might think of creating files as getting an empty bucket that you can later fill with water…or sand…or rocks…or whatever. When you create a file, you designate an empty space that you can fill with programs, activity logs, your resume, or configurations—practically anything you want, or nothing at all.

Of course, you can always create a file by writing something in an editor and saving it, as described in Chapter 4, but you will sometimes encounter situations where you just need an empty file as a placeholder for later use. You create empty files using the `touch` command, as shown in **Code Listing 2.2**.

To create a file:

1. `touch file.to.create`

   To create a file, type `touch` followed by the name of the file. This creates an empty file.


```
$ ls
$ touch file.to.create
$ ls -l file*
-rw-rw-r-        1 ejr       users       0 Jun 29 11:53 file.to.create
$ touch -t 12312359 oldfile
$ ls -l
total 0
-rw-rw-r-        1 ejr       users       0 Jun 29 11:53 file.to.create
-rw-rw-r-        1 ejr       users       0 Dec 31  2009 oldfile
$ touch -t 201012312359 new.years.eve
$ ls -l
total 0
-rw-rw-r-        1 ejr       users       0 Jun 29 11:53 file.to.create
-rw-rw-r-        1 ejr       users       0 Dec 31  2010 new.years.eve
-rw-rw-r-        1 ejr       users       0 Dec 31  2009 oldfile
$
```

Code Listing 2.2 Use the `touch` command to create files, update their modification times, or both.

2. `ls -l file*`

   Optionally, verify that the file was created by typing `ls –l file*`. As shown in Code Listing 2.2, you'll see the name of the new file as well as its length (0) and the date and time of its creation (likely seconds before the current time, if you're following along).

✔ Tips

- You can also use `touch` to update an existing file's date and time. For example, typing `touch -t 12312359 oldfile` at the prompt would update `oldfile` with a date of December 31, 23 hours, and 59 minutes in the current year. Or, typing `touch -t 201012312359 new.years.eve` would update the file called `new.years.eve` to the same time in the year 2010.
- Each time you save changes in a file, the system automatically updates the date and time. See Chapter 4 for details about editing and saving files.
- Refer to the sidebar "Naming Directories (and Files)" in this chapter for file-naming guidelines.

Copying Directories and Files with cp

When working in Unix, you'll frequently want to make copies of directories and files. For example, you may want to copy a file you're working on to keep an original, unscathed version handy. Or you might want to maintain duplicate copies of important directories and files in case you inadvertently delete them or save something over them. Accidents do happen, according to Murphy.

Whatever your reason, you copy directories and files using the `cp` command, as shown in **Code Listing 2.3**. When you copy directories and files, all you're doing is putting a duplicate in another location; you leave the original untouched.

To copy a directory:

1. `cp -r /home/ejr/Projects /home/`
   `→ shared/deb/Projects`

   At the shell prompt, type `cp -r`, followed by the old and new (to be created) directory names, to copy a complete directory. The `r` stands for "recursive," if that'll help you remember it.

2. `ls /home/shared/deb/Projects`

   You can use `ls` plus the new directory name to verify that the duplicate directory and its contents are in the intended location (Code Listing 2.3).

```
$ cp -r /home/ejr/Projects
→ /home/shared/deb/Projects
$ ls /home/shared/deb/Projects
current   new.ideas   schedule
$
```

Code Listing 2.3 Use `cp -r` to copy directories.

```
$ cp existingfile newfile
$ ls -l
total 7
-rw-rw-r- 1 ejr   users   1475 Jun 29
→ 12:18 existingfile
-rw-rw-r- 1 ejr   users   1475 Jun 29
→ 12:37 newfile
-rw-rw-r- 1 ejr   users   2876 Jun 29
→ 12:17 oldfile
$ cp -i existingfile oldfile
cp: overwrite 'oldfile'? n
$
```

Code Listing 2.4 Just use cp to copy files and add `-i` to insist that the system prompt you before you overwrite an existing file.

To copy a file:

1. `cp existingfile newfile`

   At the prompt, type `cp`, followed by the old and new (to be created) filename.

2. `ls -l`

   Optionally, check out the results with `ls -l`. The `-l` (for long format) flag displays the file sizes and dates so you can see that the copied file is exactly the same as the new one (**Code Listing 2.4**).

3. `cp -i existingfile oldfile`

   If you use `cp` with the `-i` flag, it prompts you before overwriting an existing file, also shown in Code Listing 2.4.

✔ Tips

- When copying directories and files, you can use either *absolute* (complete) names, which are measured from the root directory (`/home/ejr/Projects`), or *relative* (partial) names, which specify files or directories in relationship to the current directory (`ejr/Projects`) and aren't necessarily valid from elsewhere in the Unix file system. Using absolute names, you can manipulate directories and files with certainty anywhere in the Unix system. Using relative names, you can manipulate files only with reference to your current location.
- You can copy directories and files to or from someone else's directory. Skip to Chapter 5 to find out how to get access, then use the copying procedure described here.
- Use `cp` with a `-i` flag to force the system to ask you before overwriting files. Then, if you like that, visit Chapter 8 to find out about using aliases with `cp` so that the system always prompts you before overwriting files.
- You can compare the contents of two files or two directories using `cmp` and `dircmp`, respectively. For example, typing `cmp filename1 filename2` would compare the contents of the specified files. Use `diff` or `sdiff` to see the differences between files. See Chapter 6 for more information.

Listing Directories and Files with `ls` (More Goodies)

If you've been following along, you're probably an expert at using `ls` to list directory contents and to verify that files and directories were copied as you intended. `ls`, though, has a couple more handy uses. In particular, you can also use it to

- List filenames and information, which is handy for differentiating similar files (**Figure 2.1**).
- List all files in a directory, including hidden ones, such as `.profile` and `.login` configuration files (**Code Listing 2.5**). See Chapter 8 for more about configuration files.

To list filenames and information:

- `ls -l`

  At the shell prompt, type `ls -l` (that's a lowercase "L," not a one). You'll see the list of files in your directory fly by with the following information about each file (**Code Listing 2.6**):
  - Filename.
  - File size.
  - Date of last modification.
  - Permissions information (find out more about permissions in Chapter 5).
  - Ownership and group membership (also covered in Chapter 5).

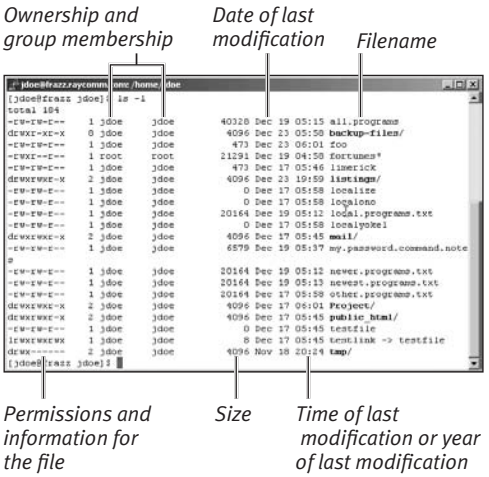

Figure 2.1 Use `ls -l` to get extra information about the directories and files in a specific directory.

```
$ ls -a
.     .stats  deb.schedule    other
..    current  new.ideas  schedule
```

Code Listing 2.5 If you want to see hidden files, use `ls -a`.

```
$ ls -l
total 13
-rw-rw-r-    1 ejr   users   2151 Jun 29 12:26 current
-rw-rw-r-    2 ejr   users   1475 Jun 29 12:35 deb.schedule
-rw-rw-r-    1 ejr   users   4567 Jun 29 12:26 new.ideas
drwxrwxr-x   2 ejr   users   1024 Jun 29 13:06 other
-rw-rw-r-    1 ejr   users   1475 Jun 29 12:22 schedule
```

Code Listing 2.6 Use `ls -l` to see a listing of the contents of a directory in long format.

- Time of last modification (if the file's been modified recently) or year of last modification (if the file was last modified more than six months previously). Check out `touch` earlier in this chapter to see how files might have modification dates in the future.

To list all files in a directory:

- `ls -la`

  Enter `ls -la` at the shell prompt to list all the files in the directory, including hidden files, with full information, as shown in **Code Listing 2.7**.

✔ Tips

- You can hide files by giving them a name that starts with a dot (`.`). That is, `profile` would not be hidden, but `.profile` would be.
- Remember, you can combine any flags to specify multiple options. For example, if you want to list all files (`-a`) in the long format (`-l`) you would use `ls -la`.
- Try `ls -ltR` to get the complete listing of your current directory, the directories it contains, and so forth until you run out of subdirectories to descend into.


```
$ ls -la
total 22
drwxrwxr-x   3 ejr   users   1024 Jun 29 13:07 .
drwxrwx--    7 ejr   users   1024 Jun 29 12:16 ..
-rw-rw-r-    1 ejr   users   6718 Jun 29 13:00 .stats
-rw-rw-r-    1 ejr   users   2151 Jun 29 12:26 current
-rw-rw-r-    2 ejr   users   1475 Jun 29 12:35 deb.schedule
-rw-rw-r-    1 ejr   users   4567 Jun 29 12:26 new.ideas
drwxrwxr-x   2 ejr   users   1024 Jun 29 13:06 other
-rw-rw-r-    1 ejr   users   1475 Jun 29 12:22 schedule
$
```

Code Listing 2.7 If you want to see everything, use `ls -la`.

Moving Files with mv

Moving directories and files means moving them from one location (think of *location* as an absolute file path, like `/home/ejr/aFile`) in your system to another location (say, `/tmp/File` or `/home/ejr/AnotherFile`). Essentially, you have only one version of a file, and you change the location of that version. For example, you might move a directory when you're reorganizing your directories and files. Or, you might move a file to rename it—that is, move a file from one name to another name.

You move directories and files using `mv`, as shown in **Code Listing 2.8**.

```
$ ls
Completed     existingfile    oldfile
$ mv existingfile newfile
$ ls
Completed     newfile         oldfile
$
```

Code Listing 2.8 List files to see the current files, then use `mv` to rename one of the files.

To move a file or directory:

1. `ls`

   To begin, use `ls` to verify the name of the file you want to move. If you're changing the name of the file, you'll want to ensure that the new filename isn't yet in use. If you move a file to an existing filename, the contents of the old file will be replaced with the contents of the new file.

2. `mv existingfile newfile`

   Type `mv` plus the existing filename and the new filename. Say goodbye to the old file and hello to the new one (Code Listing 2.8).

   You use the same process—exactly—to move directories; just specify the directory names, as in `mv ExistingDirectory NewDirectory`.

3. `ls`

   Verify that the file is now located in the location you intended.

✔ Tips

- You can also use `mv` to move files into or out of directories. For example, `mv Projects/temp/testfile /home/deb/testfile` moves `testfile` from the `Projects` and `temp` subdirectories of the current directory to Deb's home directory, also using the name `testfile`.
- Use `mv -i oldfilename newfilename` to require the system to prompt you before overwriting (destroying) existing files. The `-i` is for "interactive," and it also works with the `cp` command.
- Check out Chapter 8 to learn how to use aliases with `mv` so that the system always prompts you before overwriting files and you don't have to remember the `-i` flag.
- If you use `mv` and specify an existing directory as the target (as in, `mv something ExistingDirectory`), "something," in this case, will be placed into `ExistingDirectory`. "Something" can be either a file or a directory.

```
$ ls
Completed              oldfile
newfile                soon.to.be.gone.file
$ rm -i soon.to.be.gone.file
rm: remove    'soon.to.be.gone.file'? y
$ ls
Completed     newfile oldfile
$
```

Code Listing 2.9 Use `rm -i` to safely and carefully remove directories and files.

Removing Files with `rm`

You can easily—perhaps too easily—remove (delete) files from your Unix system. As Murphy will tell you, it's a good idea to think twice before doing this; once you remove a file, it's gone (unless, of course, you plead with your system administrator to restore it from a backup tape—but that's another story). At any rate, it's permanent, unlike deletions in Windows or Mac OS, or even many Unix desktop environments like GNOME or KDE, where the Recycle Bin or Trash give you a second chance.

You remove files using `rm`, as shown in **Code Listing 2.9**. And, as you'll see in the following steps, you can remove files one at a time or several at a time.

To remove a file:

1. `ls -l`

   List the files in the current folder to verify the name of the file you want to remove.

2. `rm -i soon.to.be.gone.file`

   At your shell prompt, type `rm -i` followed by the name of the file you want to remove. The `-i` tells the system to prompt you before removing the files (Code Listing 2.9).

3. `ls`

   It is gone, isn't it?

To remove multiple files:

1. `ls -l *.html`

   List the files to make sure you know which files you want to remove (and not remove).

2. `rm -i *.html`

   Using the asterisk wildcard (`*`), you can remove multiple files at one time. In this example, we remove all files in the current directory that end with `.html`. (Refer to Chapter 1, specifically the section called "Using Wildcards," for details about using wildcards.)

   or

1. `rm -i dangerous`

   Here, `-i` specifies that you'll be prompted to verify the removal of a directory or file named `dangerous` before it's removed.

2. `rm -ir dan*`

   This risky command removes all of the directories or files that start with `dan` in the current directory and all of the files and subdirectories in the subdirectories starting with `dan`. If you're sure, don't use the `-i` flag to just have the files removed without prompting you for confirmation. (Remember that the flags `-ir` could also be written as `-i -r` or `-ri` or `-r -i`. Unix is rather flexible.)

Can You Really Screw Up the System?

In general, no. When you log in to a Unix system and use your personal userid, the worst you can do is remove your own directories and files. As long as you're logged in as yourself, commands you type won't affect anything critical to the Unix system, only your own personal directories and files. Score one for Unix—as an average user, you cannot really break the system. With Windows, though, it can be a different story.

If you have system administrator rights, meaning that you can log in as `root` (giving you access to all the system directories and files), you can do a lot of damage if you're not extremely careful. For this reason, don't log in as `root` unless you absolutely have to.

Many newer systems won't even let you log in as `root`. Instead, you need to use `su` or an equivalent, as discussed in Chapter 3. There, you'll also find more information about `su`, which can help reduce the risk of being logged in as `root`.

✔ Tips

- If you have system administrator rights (or are logged in as `root`, rather than with your userid), be extremely careful when using `rm`. Rather than remove merely your personal directories or files, you could potentially remove system directories and files. Scope out the sidebar "Can You Really Screw Up the System?"
- This is a good time to remind you to use the handy `cp` command to make backup copies of anything you value—before you experiment too much with `rm`. Even if the system administrator keeps good backups, it's ever so much easier if you keep an extra copy of your goodies sitting around. Try `cp –r . backup_files` for a space-hogging—but effective—means of making a quick backup of everything in the current directory into the `backup-files` directory. (Just ignore the error message about not copying a directory into itself-the system will do the right thing for you, and you don't have to worry about it.)
- We suggest using `rm -i`, at least until you're sure you're comfortable with irrevocable deletions. The `-i` flag prompts you to verify your command before it's executed.
- Check out Chapter 8 to find out about using aliases with `rm` so that the system always prompts you before removing the directories or files even if you forget the `-i` flag.
- If you accidentally end up with a file that has a problematic filename (like one that starts with `-`, which looks to Unix like a command flag, not a filename), you can delete it (with a trick). Use `rm –i -- bad-filename` to get rid of it.

Removing Directories with `rmdir`

Another handy thing you can do is remove directories using `rmdir`. Think of removing directories as trimming branches on a tree. That is, you can't be sitting on the branch you want to trim off. You have to sit on the next closest branch; otherwise, you'll fall to the ground along with the branch you trim off. Ouch! Similarly, when you remove a directory, you must not be located in the directory you want to remove.

You must remove a directory's contents (all subdirectories and files) before you remove the directory itself. In doing so, you can verify what you're removing and avoid accidentally removing important stuff. In the following steps (illustrated in **Code Listing 2.10**), we'll show you how to remove a directory's contents, and then remove the directory itself.


```
$ cd /home/ejr/Yourdirectory
$ ls -la
total 7
drwxrwxr-x       2 ejr       users       1024 Jun 29 20:59 .
drwxrwx--        8 ejr       users       1024 Jun 29 20:59 ..
-rw-rw-r-        1 ejr       users       1475 Jun 29 20:59 cancelled.project.notes
-rw-rw-r-        1 ejr       users       2876 Jun 29 20:59 outdated.contact.info
$ rm *
$ cd ..
$ rmdir Yourdirectory
$ ls
Newdirectory     all.programs.txt       newer.programs          short.fortunes
Projects         files                  newest.programs         temp
Xrootenv.0       fortunes               newstuff                touching
all.programs  local.programs.txt  schedule
$
```

Code Listing 2.10 Removing directories with `rmdir` can be a little tedious—but better safe than sorry.

✔ Tips

- You can remove multiple directories at one time. Assuming you're starting with empty directories, just list them like this: `rmdir Yourdirectory Yourotherdirectory OtherDirectory`
- As an alternative to `rmdir`, you can remove a directory and all of its contents at once using `rm` with the `-r` flag; for example, `rm -r Directoryname`. Be careful, though! This method automatically removes the directory and everything in it, so you won't have the opportunity to examine everything you remove beforehand. If you're getting asked for confirmation before deleting each file and you're really, absolutely, positively, completely sure that you're doing the right thing, use `rm –rf Directoryname` to force immediate deletion.
- If you're getting comfortable with long command strings, you can specify commands with a complete directory path, as in `ls /home/ejr/DirectorytoGo` or `rm /home/ejr/DirectorytoGo/*`. This technique is particularly good if you want to be absolutely sure that you're deleting the right directory, and not a directory with the same name in a different place on the system.

To remove a directory:

1. `cd /home/ejr/Yourdirectory`

   To begin, change to that directory by typing `cd` plus the name of the directory you want to remove.

2. `ls -a`

   List all (`-a`) of the files, including any hidden files that might be present, in the directory, and make sure you don't need any of them. If you see only `.` and `..` (which indicate the current directory and its parent directory, respectively), you can skip ahead to step 4.

3. Do one or both of these:
   - If you have hidden files in the directory, type `rm .* *` to delete those files plus all of the rest of the files.
   - If you have subdirectories in the directory, type `cd` and the subdirectory name, essentially repeating the process starting with step 1. Repeat this process until you remove all subdirectories.

   When you finish this step, you should have a completely empty directory, ready to be removed.

4. `cd ..`

   Use the change directory command again to move up one level, to the parent of the directory that you want to remove.

5. `rmdir Yourdirectory`

   There it goes—wave goodbye to the directory! See Code Listing 2.10 for the whole sequence.

Finding Forgotten Files with `find`

Where, oh where, did that file go? Sometimes finding a file requires more than cursing at your computer or listing directory contents with `ls`. Instead, you can use the `find` command, which lets you search in dozens of ways, including through the entire directory tree (**Code Listing 2.11**) or through directories you specify (**Code Listing 2.12**).

To find a file:

- `find . -name lostfile -print`

  Along with the `find` command, this specifies to start in the current directory with a dot (`.`), provide the filename (`-name lostfile`), and specify that the results be printed onscreen (`-print`). See Code Listing 2.11.

To find files starting in a specific directory:

- `find /home/deb -name 'pending*' -print`

  This command finds all of the files and directories with names starting with `pending` under Deb's home directory. You must use single quotes if you include a wildcard to search for.

Or, you can find files under multiple directories at one time, like this:

- `find /home/deb /home/ejr -name 'pending*' -print`

  This command finds files with names starting with `pending` in Deb's and Eric's home directories or any subdirectories under them (Code Listing 2.12).

```
$ find . -name lostfile -print
./Projects/schedule/lostfile
$
```

Code Listing 2.11 Use `find` to locate a missing file.

```
$ find /home/deb -name 'pending*' -print
/home/deb/Projects/schedule/pending.tasks
$ find /home/deb /home/ejr -name
→ 'pending*' -print
/home/deb/Projects/schedule/pending.tasks
/home/ejr/pending.jobs.to.do.today.to.do
$
```

Code Listing 2.12 By using wildcards and specifying multiple directories, you can make `find` yet more powerful.

To find and act on files:

- `find ~ -name '*.backup' -ok rm {} \;`
  Type `find` with a wildcard expression, followed by `-ok` (to execute the following command, with confirmation), `rm` (the command to issue), and `{} \;` to fill in each file found as an *argument* (an additional piece of information) for the command. If you want to, say, compress matching files without confirmation, you might use `find ~ -name '*.backup' -exec compress {} \;` to do the work for you.

✔ Tips

- On some Unix systems, you may not need the `-print` flag. Try entering `find` without the `-print` flag. If you see the results onscreen, then you don't need to add the `-print` flag.
- Avoid starting the `find` command with the root directory, as in `find / -name the.missing.file -print`. In starting with the root directory (indicated by the `/`), you'll likely encounter a pesky error message for each directory you don't have access to, and there will be a lot of those. Of course, if you're logged in as `root`, feel free to start with `/`.
- If you know only part of the filename, you can use quoted wildcards with `find`, as in `find . -name '*info*' -print`.
- `find` offers many chapters' worth of options. If you're looking for a specific file or files based on any characteristics, you can find them with `find`. For example, you can use `find /home/shared -mtime -3` to find all files under the shared directory that were modified within the last three days. See Appendix C for a substantial (but not comprehensive) listing of options.

Locating Lost Files with `locate`

If you're looking for a system file—that is, a program or file that is part of the Unix system itself, rather than one of your personal files in your home directory—try `locate` to find it. You'll get more results than you can handle, but it's a quick and easy way to locate system files.

The `locate` command isn't available on all Unix systems, but it is worth a try at any rate. See **Code Listing 2.13** for `locate` in action.

To locate a file:

- `locate fortune`

  If you try to locate `fortune`, you'll get a listing of all of the system files that contain "fortune" in them. This listing includes the `fortune` program, fortune data files for the fortune program to use, and related stuff. It's a huge list in most cases (Code Listing 2.13).

✔ Tips

- Use `locate` in combination with `grep` (see "Using Regular Expressions with `grep`" in Chapter 6) to narrow down your list, if possible.
- Many people use `locate` to get a quick look at the directories that contain relevant files (`/usr/share/games/fortunes` contains a lot of files related to the fortune program), then other tools to take a closer look.
- Not all systems include fortune—it's certainly just a fun thing and not essential by any means. If you don't "locate" it, try looking for `bash` or `zsh` (known as shells) to see how `locate` works. (See Chapter 8 for more information about different shells and their benefits and drawbacks.)

```
[jdoe@frazz jdoe]$ locate fortune
/usr/share/man/man6/fortune.6.bz2
/usr/share/doc/fortune-mod-1.0
/usr/share/doc/fortune-mod-1.0/cs
/usr/share/doc/fortune-mod-1.0/cs/HISTORIE
/usr/share/doc/fortune-mod-1.0/cs/LICENSE
/usr/share/doc/fortune-mod-1.0/cs/README
/usr/share/doc/fortune-mod-1.0/fr
/usr/share/doc/fortune-mod-1.0/fr/
→ COPYING.linuxfr
/usr/share/doc/fortune-mod-1.0/fr/
→ COPYING.glp
/usr/share/doc/fortune-mod-1.0/fr/ffr
...
/usr/share/games/fortunes/songs-poems
/usr/share/games/fortunes/sports.dat
/usr/share/games/fortunes/sports
/usr/share/games/fortunes/startrek.dat
/usr/share/games/fortunes/startrek
/usr/share/games/fortunes/translate-me.dat
/usr/share/games/fortunes/translate-me
/usr/share/games/fortunes/wisdom.dat
/usr/share/games/fortunes/wisdom
/usr/share/games/fortunes/work.dat
/usr/share/games/fortunes/work
/usr/share/games/fortunes/zippy.dat
/usr/share/games/fortunes/zippy
/usr/share/sol-games/fortunes.scm
/usr/games/fortune
[jdoe@frazz jdoe]$
```

Code Listing 2.13 Use `locate` to find everything—everything—related to most system files.

Linking with `ln` (Hard Links)

Suppose your boss just hired an assistant for you ('bout time, right?). You'll need to make sure your new helper can access your files so you can pawn off your work on him. And you'll need to access the revised files just so you can keep up with what your helper's been doing—and perhaps take credit for his work at the next staff meeting.

A great way to give your helper easy access to your files is to create a *hard link* from your home directory. In making a hard link, all you're doing is starting with an existing file and creating a link, which (sort of) places the existing file in your helper's home directory. The link does not create a copy of the file; instead, you're creating a second pointer to the same physical file on the disk. Rather than the additional pointer being secondary (like an alias or shortcut in Macintosh or Windows computers), both of the pointers reference the same actual file, so from the perspective of the Unix system, the file actually resides in two locations (**Code Listing 2.14**).

Because using hard links often requires that you have access to another user's home directory, you might venture to Chapter 5 for details about using `chmod`, `chgrp`, and `chown` to access another user's directories and files.


```
$ ls /home/deb/Projects/schedule/our* /home/helper/our*
ls: /home/helper/our*: No such file or directory
/home/deb/Projects/schedule/our.projects.latest
/home/deb/Projects/schedule/our.projects.other
$ ln /home/deb/Projects/schedule/our.projects.latest /home/helper/our.projects
$ ls -l /home/helper/o*
-rw-r-r-     3 ejr       users       1055 Jun 26 11:00 /home/helper/our.projects
$
```

Code Listing 2.14 Hard links let two users easily share files.

To make a hard link:

1. `ls -l /home/deb/Projects/schedule/ → our* /home/helper/our*`

   To begin, list the files in both directories to make sure that the file to link exists and that there's no other file with the intended name in the target directory. Here, we list the files that start with `our` in both `/home/deb/Projects/schedule` and in `/home/helper`. In this example, we're verifying that the file does exist in Deb's directory and that no matching files were found in the helper's directory (Code Listing 2.14).

2. `ln /home/deb/Projects/schedule/ → our.projects.latest → /home/helper/our.projects`

   Here, `ln` creates a new file with a similar name in the helper's home directory and links the two files together, essentially making the same file exist in two home directories.

3. `ls -l /home/helper/o*`

   With this code, your helper can verify that the file exists by listing files that begin with `o`.

   Now the file exists in two places with exactly the same content. Either user can modify the file, and the content in both locations will change.

✔ Tips

- You can remove hard links just like you remove regular files, by using `rm` plus the filename. See the section "Removing Files with `rm`," earlier in this chapter.
- If one user removes the file, the other user can still access the file from his or her directory.
- Hard links work from file to file only within the same file system. To link directories or to link across different file systems, you'll have to use soft links, which are covered in the next section.
- If you're sneaky, you can use hard links to link directories, not just files. Make a new directory where you want the linked directory to be, and then use `ln /home/whoever/existingdirectory/* /home/you/newdirectory/` to hard-link all of the files in the old directory to the new directory. New files won't be linked automatically, but you could use a `cron` job to refresh the links periodically—say, daily. See Chapter 9 for `cron` details.

Linking with `ln -s` (Soft Links)

Now suppose you want to pawn off your entire workload on your new helper. Rather than just give him access to a single file, you'll want to make it easy for him to access your entire project directory. You can do this using soft links (created with `ln -s`), which essentially provide other users with a shortcut to the file or directory you specify.

Like hard links, *soft links* allow a file to be in more than one place at a time; however, with soft links, there's only one copy of it and, with soft links, you can link directories as well. The linked file or directory is dependent on the original one—that is, if the original file or directory is deleted, the linked file or directory will no longer be available. With hard links, the file is not actually removed from disk until the last hard link is deleted.

Soft links are particularly handy because they work for directories as well as individual files, and they work across different file systems (that is, not just within `/home`, but anywhere on the Unix system).

Like hard links, soft links sometimes require that you have access to another user's directory and files. See Chapter 5 for more on file permissions and ownership and Chapter 7 for the lowdown on file systems.

To make a soft link:

1. `ls /home/deb /home/helper`

   To begin, list the contents of both users' home directories. Here, we're verifying that the directory we want to link does exist in Deb's directory and that no matching directories or files exist in the helper's directory. See **Code Listing 2.15**.

2. `ln -s /home/deb/Projects`
   `→ /home/helper/Projects`

   This command creates a soft link so the contents of Deb's Projects directory can also be easily accessed from the helper's home directory.

3. `ls -la /home/helper`

   Listing the contents of `/home/helper` shows the existence of the soft link to the directory. Notice the arrow showing the link in Code Listing 2.15.

✔ Tip

- If you only need to create a link between two files within the same file system, consider using hard links, as discussed in the previous section, "Linking with `ln` (Hard Links)."


```
$ ls /home/deb /home/helper
/home/deb:
Projects
/home/helper:
our.projects
$ ln -s /home/deb/Projects /home/helper/Projects
$ ls -la /home/helper/
total 11
d-wxrwx--        2 helper        users       1024 Jun 29 21:18 .
drwxr-xr-x       11 root         root        1024 Jun 29 21:03 ..
-rw-rwxr-        1 helper        users       3768 Jun 29 21:03 .Xdefaults
-rw-rwxr-        1 helper        users       24 Jun 29 21:03 .bash_logout
-rw-rwxr-        1 helper        users       220 Jun 29 21:03 .bash_profile
-rw-rwxr-        1 helper        users       124 Jun 29 21:03 .bashrc
lrwxrwxrwx       1 ejr           users       18 Jun 29 21:18 Projects -> /home/deb/Projects
-rw-rwxr-        3 ejr           users       1055 Jun 26 11:00 our.projects
$
```

Code Listing 2.15 Use `ln -s` to make soft links and connect directories.