VISUAL QUICKSTART GUIDE

UNIX AND LINUX

FOURTH EDITION

Learn Unix and Linux the Quick and Easy Way!

DEBORAH S. RAY
ERIC J. RAY

Table of Contents

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

2 Using Directories and Files

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

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

```
$ 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

- 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.
- 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 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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

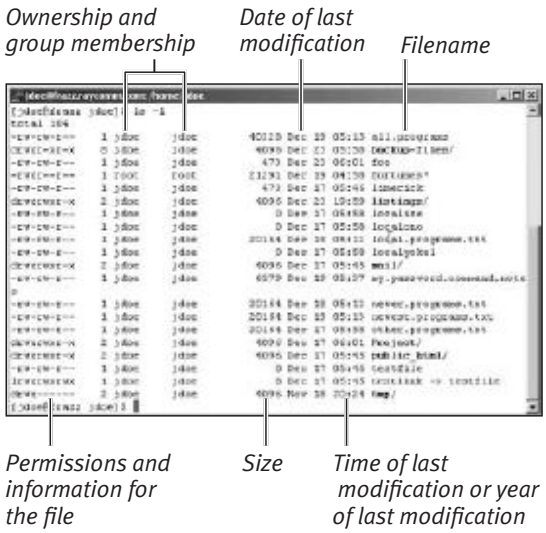

Figure 2.1 Use `ls -l` to get extra information about the directories and files in a specific directory.

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).

```
$ 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`.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

```
$ 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`.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

✔ 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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

✔ 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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893

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.

Chapter 2. Using Directories and Files. Unix and Linux: Visual QuickStart Guide, Fourth Edition, ISBN: 9780321657893