HOUR 3

Moving About the File System

This third hour focuses on the Unix hierarchical file system. You learn how the system is organized, how it differs from the Macintosh and Windows hierarchical file systems, the difference between relative and absolute filenames, and the mysterious `.` and `..` directories. You also learn about the `env`, `pwd`, and `cd` commands and the `HOME` and `PATH` environment variables.

Goals for This Hour

In this hour, you will learn

- What a hierarchical file system is all about
- How the Unix file system is organized
- How Mac and PC file systems differ from Unix
- The difference between relative and absolute filenames
- About hidden files in Unix
- About the special directories `.` and `..`
- About the `env` command
- About user environment variables, `PATH` and `HOME`
- How to find where you are with `pwd`
- How to move to another location with `cd`

The preceding hour introduced many Unix commands, but this hour takes a more theoretical approach, focusing on the Unix file system, how it's organized, and how you can navigate it. This hour focuses on the environment that tags along with you as you move about, particularly the `HOME` and `PATH` variables. After that is explained, you learn about the `env` command as an easy way to show environment variables, and you learn the `pwd` and `cd` pair of commands for moving about directly.

What a Hierarchical File System Is All About

In a nutshell, a hierarchy is a system organized by graded categorization. A familiar example is the organizational structure of a company, where workers report to supervisors and supervisors report to middle managers. Middle managers, in turn, report to senior managers, and senior managers report to vice-presidents, who report to the president of the company. Graphically, this hierarchy looks as shown in Figure 3.1.

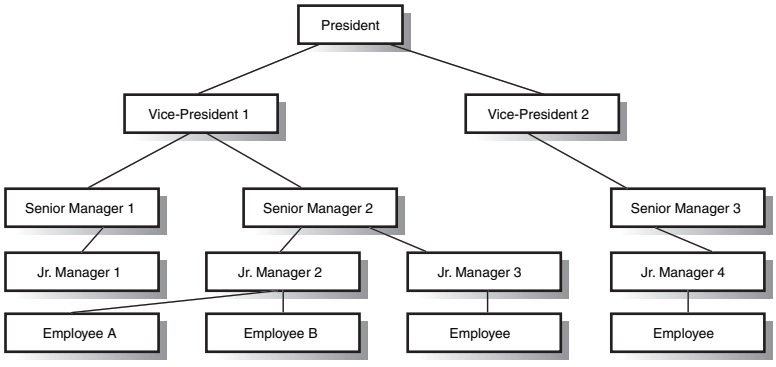

FIGURE 3.1
A typical organizational hierarchy.

You've doubtless seen this type of illustration before, and you know that a higher position indicates more control. Each position is controlled by the next highest position or row. The president is top dog of the organization, but each subsequent manager is also in control of his or her own small fiefdom.

To understand how a file system has a similar organization, imagine each of the managers in the illustration as a file folder and each of the employees as a piece of paper, filed in a particular folder. Open any file cabinet, and you probably see things organized this way: Filed papers are placed in labeled folders, and often these folders are filed in groups under specific topics. The drawer might then have a specific label to distinguish it from other drawers in the cabinet, and so on.

That's exactly what a hierarchical file system is all about. You want to have your files located in the most appropriate place in the file system, whether at the very top, in a folder, or in a nested series of folders. With careful usage, a hierarchical file system can contain thousands of files and still allow users to find any individual file quickly.

On my computer, the chapters of this book are organized in a hierarchical fashion, as shown in Figure 3.2.

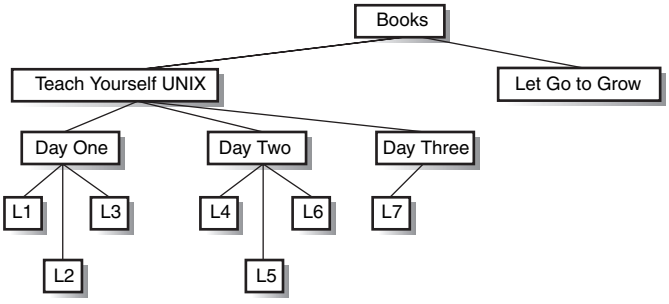

FIGURE 3.2
File organization for the chapters of *Sams Teach Yourself Unix in 24 Hours, Fourth Edition*.

Task 3.1: The Unix File System Organization

A key concept enabling the Unix hierarchical file system to be so effective is that anything that is not a folder is a file. Programs are files in Unix, device drivers are files, documents and spreadsheets are files, your keyboard is represented as a file, your display is a file, and even your tty line and mouse are files.

What this means is that as Unix has developed, it has avoided becoming an ungainly mess. Unix does not have hundreds of cryptic files stuck at the top (this is still a problem in DOS) or tucked away in confusing folders within the System Folder (as with the Macintosh).

The top level of the Unix file structure (/) is known as the *root* directory or *slash* directory, and it always has a certain set of subdirectories, including bin, dev, etc, lib, mnt, tmp, and usr. There can be a lot more, however.

You can obtain a listing of the files and directories in your own top-level directory by using the `ls -F /` command. (You'll learn all about the `ls` command in the next hour. For now, just be sure that you enter exactly what's shown in the example.)

On a different computer system, here's what I see when I enter that command:

```
% ls -F /
Mail/         export/         public/
News/         home/           reviews/
add_swap/     kadb*           sbin/
apps/         layout          sys@
archives/     lib@            tftpboot/
bin@          lost+found/     tmp/
boot          mnt/            usr/
cdrom/        net/            utilities/
chess/        news/           var/
dev/          nntpserver      vmunix*
etc/          pcfs/
```

In this example, any filename that ends with a slash (/) is a folder (Unix calls these *directories*). Any filename that ends with an asterisk (*) is a program. Anything ending with the at sign (@) is a *symbolic link* (a pointer to another file or directory elsewhere in the file system), and everything else is a normal, plain file.

As you can see from this example, and as you'll immediately find when you try the command yourself, there is much variation in how different Unix systems organize the top-level directory. There are some directories and files in common, and once you start examining the contents of specific directories, you'll find that hundreds of programs and files always show up in the same place from Unix to Unix.

It's as if you were working as a file clerk at a new law firm. Although this firm might have a specific approach to filing information, the approach can be similar to the filing system of other firms where you have worked in the past. If you know the underlying organization, you can quickly pick up the specifics of a particular organization.

1. Try the command `ls -F /` on your computer system, and identify, as previously explained, each of the directories in your resultant listing.

The output of the previous `ls` command shows the files and directories in the top level of your system. Next, you learn what the commonly found directories are.

The `bin` Directory

In Unix parlance, programs are considered *executables* because users can execute them. (In this case, *execute* is a synonym for *run*, not an indication that you get to wander about murdering innocent applications!) When the program has been compiled, it is translated from source code into what's called a *binary* format. Add the two together, and you have a common Unix description for an application—an executable binary.

It's no surprise that the original Unix developers decided to have a directory labeled `binaries` to store all the executable programs on the system. Remember the primitive teletypewriter discussed earlier? Having a slow system to talk with the computer had many ramifications you might not expect. The single most obvious one was that everything became quite concise. There were no lengthy words like `binaries` or `listfiles`, but rather succinct abbreviations: `bin` and `ls` are, respectively, the Unix equivalents.

The `bin` directory (pronounce it to rhyme with "tin") is where all the executable binaries were kept in early Unix. Over time, as more and more executables were

added to Unix, having all the executables in one place proved unmanageable, and the `bin` directory split into multiple parts (`/bin`, `/sbin`, `/usr/bin`).

The `dev` Directory

Among the most important portions of any computer are its device drivers. Without them, you wouldn't have any information on your screen (the information arrives courtesy of the display device driver). You wouldn't be able to enter information (the information is read and given to the system by the keyboard device driver), and you wouldn't be able to use your floppy disk drive (managed by the floppy device driver).

Remember, everything in Unix is a file. Every component of the system, from the keyboard driver to the hard disk, is a file.

Earlier, you learned how almost anything in Unix is considered a file in the file system, and the `dev` directory is an example. All device drivers—often numbering into the hundreds—are stored as separate files in the standard Unix `dev` (devices) directory. Pronounce this directory name "dev," not "dee-ee-vee."

The `etc` Directory

Unix administration can be quite complex, involving management of user accounts, the file system, security, device drivers, hardware configurations, and more. To help, Unix designates the `etc` directory as the storage place for all administrative files and information.

Pronounce the directory name "ee-tea-sea," "et-sea," or "etcetera." All three pronunciations are common.

The `lib` Directory

Like your own community, Unix has a central storage place for function and procedural libraries. These specific executables are included with specific programs, allowing programs to offer features and capabilities otherwise unavailable. The idea is that if programs want to include certain features, they can reference only the shared copy in the Unix library rather than having a new, unique copy.

Pronounce the directory name "libe" or "lib" (to rhyme with the word *bib*).

The `lost+found` Directory

With multiple users running many different programs simultaneously, it's been a challenge over the years to develop a file system that can remain synchronized with the activity of the computer. Various parts of the Unix *kernel*—the brains of the

system—help with this problem. When files are recovered after any sort of problem or failure, they are placed here, in the `lost+found` directory, if the kernel cannot ascertain the proper location in the file system. This directory should be empty almost all the time.

This directory is commonly pronounced "lost and found" rather than "lost plus found."

The `mnt` and `sys` Directories

The `mnt` (pronounced "em-en-tea") and `sys` (pronounced "sis") directories are safely ignored by Unix users. The `mnt` directory is intended to be a common place to mount external media—hard disks, removable cartridge drives, and so on—in Unix. On many systems, though not all, `sys` contains files indicating the system configuration.

The `tmp` Directory

A directory that you can't ignore, the `tmp` directory—say "temp"—is used by many of the programs in Unix as a temporary file-storage space. If you're editing a file, for example, the editor makes a copy of the file, saves it in `tmp`, and you work directly with that, saving the new file back to your original only when you've completed your work.

On most systems, `tmp` ends up littered with various files and executables left by programs that don't remove their own temporary files. On one system I use, it's not uncommon to find 10–30 megabytes of files wasting space.

Even so, if you're manipulating files or working with copies of files, `tmp` is the best place to keep the temporary copies. Indeed, on some Unix workstations, `tmp` actually can be the fastest device on the computer, allowing for dramatic performance improvements over working with files directly in your home directory.

The `usr` Directory

The last of the standard directories at the top level of the Unix file system hierarchy is the `usr`—pronounced "user"—directory. Originally, this directory was intended to be the central storage place for all user-related commands. Today, however, many companies have their own interpretation, and there's no telling what you'll find in this directory.

Other Miscellaneous Stuff at the Top Level

In addition to all the directories previously listed, various other directories and files commonly occur in Unix systems. Some files might have slight variations in name on your computer, so when you compare your listing to the following files and directories, be alert for possible alternative spellings.

A file you must have in order to bring up Unix at all is one usually called unix or `vmunix`, or named after the specific version of Unix on the computer. The file contains the actual Unix operating system. The file must have a specific name and must be found at the top level of the file system. Hand-in-hand with the operating system is another file called `boot`, which helps during initial startup of the hardware.

Notice on one of the previous listings that the files `boot` and `dynix` appear. (DYNIX is the name of the particular variant of Unix used on Sequent computers.) By comparison, the listing from the Sun Microsystems workstation shows `boot` and `vmunix` as the two files.

Another directory you might find in your own top-level listing is `diag`—pronounced "dye-ag"—which acts as a storehouse for diagnostic and maintenance programs. If you have any programs within this directory, it's best not to try them out without proper training!

The *home directory*, `/home`, also sometimes called *users*, is a central place for organizing all files owned by specific users. Listing this directory is usually an easy way to find out what accounts are on the system, too, because by convention each individual account directory is named after the user's account name. On one system I use, my account is `taylor`, and my individual account directory is also called `taylor`. Home directories are always created by the system administrator.

The `net` directory, if set up correctly, is a handy shortcut for accessing other computers on your network.

The `tftpboot` directory is a relatively new feature of Unix. The letters stand for "trivial file transfer protocol boot." Don't let the name confuse you, though; this directory contains versions of the kernel suitable for X Window System–based terminals and diskless workstations to run Unix.

Some Unix systems have directories named for specific types of peripherals that can be attached. On the Sun workstation, you can see examples with the directories `cdrom` and `pcfs`. The former is for a CD-ROM drive and the latter for DOS-format floppy disks.

Many more directories are in Unix, but this will give you an idea of how things are organized.

Directory Separator Characters

If you look at the organizational chart presented earlier in this hour (refer to Figure 3.1), you see that employees are identified simply as "employee" where possible. Because each has a unique path upward to the president, each has a unique identifier if all components of the path upward are specified.

For example, the rightmost of the four employees could be described as "Employee managed by Jr. Manager 4, managed by Senior Manager 3, managed by Vice-President 2, managed by the President." Using a single character, instead of "managed by," can considerably shorten the description: Employee/Jr. Manager 4/Senior Manager 3/Vice-President 2/President. Now consider the same path specified from the very top of the organization downward: President/Vice-President 2/Senior Manager 3/Jr. Manager 4/Employee.

Because only one person is at the top, that person can be safely dropped from the path without losing the uniqueness of the descriptor: /Vice-President 2/Senior Manager 3/Jr. Manager 4/Employee.

In this example, the / (pronounce it "slash") is serving as a *directory separator character*, a convenient shorthand to indicate different directories in a path.

The idea of using a single character isn't unique to Unix, but using the slash is unusual. On the Macintosh, the system uses a colon to separate directories in a pathname. (Next time you're on a Mac, try saving a file called `test:file` and see what happens.) DOS uses a backslash: `\DOS` indicates the `DOS` directory at the top level. The characters `/tmp` indicate the `tmp` directory at the top level of the Unix file system, and `:Apps` is a folder called `Apps` at the top of the Macintosh file system.

On the Macintosh, you rarely encounter the directory delineator because almost all users live in the graphical interface and don't even know that there's a Unix system—and command line interface—lurking beneath the Aqua environment. Windows also offers a similar level of freedom from having to worry about much of this complexity, although you'll still need to remember whether "A:" is your floppy disk or hard disk drive.

The Difference Between Relative and Absolute Filenames

Specifying the exact location of a file in a hierarchy to ensure that the filename is unique is known in Unix parlance as specifying its *absolute filename*. That is, regardless of where you are within the file system, the absolute filename always specifies a particular file. By contrast, relative filenames are not unique descriptors.

To understand, consider the files shown in Figure 3.3.

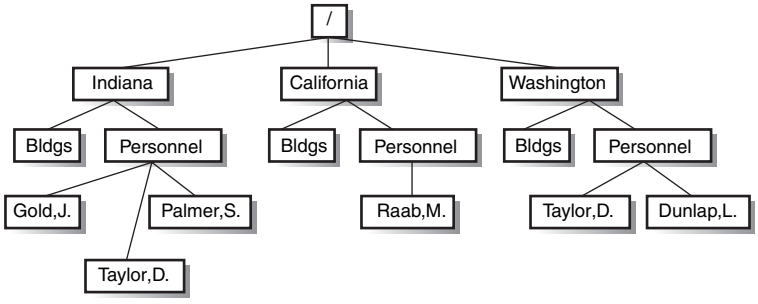

FIGURE 3.3
A simple hierarchy of files.

If you are currently looking at the information in the `Indiana` directory, `Bldgs` uniquely describes one file: the `Bldgs` file in the `Indiana` directory. That same name, however, refers to a different file if you are in the `California` or `Washington` directories. Similarly, the directory `Personnel` leaves you with three possible choices until you also specify which state you're interested in.

As a possible scenario, imagine you're reading through the `Bldgs` file for `Washington` and some people come into your office, interrupting your work. After a few minutes of talk, they comment about an entry in the `Bldgs` file in `California`. You turn to your Unix system and bring up the `Bldgs` file, and it's the wrong file. Why? You're still in the `Washington` directory.

These problems arise because of the lack of specificity of *relative filenames*. Relative filenames describe files that are referenced relative to an assumed position in the file system. In Figure 3.3, even `Personnel/Taylor,D.` isn't unique because that can be found in both `Indiana` and `Washington`.

To avoid these problems, you can apply the technique you learned earlier, specifying all elements of the directory path from the top down. To look at the `Bldgs` file for `California`, you could simply specify `/California/Bldgs`. To check the `Taylor,D.` employee in `Indiana`, you'd use `/Indiana/Personnel/Taylor,D.`, which is different, you'll notice, from `/Washington/Personnel/Taylor,D.`.

Learning the difference between these two notations is crucial to surviving the complexity of the hierarchical file system used with Unix. Without it, you'll spend half your time verifying that you are where you think you are, or, worse, not moving about at all, not taking advantage of the organizational capabilities.

If you're ever in doubt as to where you are or what file you're working with in Unix, simply specify its absolute filename. You always can differentiate between the two

by looking at the very first character: If it's a slash, you've got an absolute filename (because the filename is rooted to the very top level of the file system). If you don't have a slash as the first character, the filename's a relative filename.

Earlier I told you that in the `/home` directory at the top level of Unix, I have a home directory called `taylor`. In absolute filename terms, I'd properly say that I have `/home/taylor` as a unique directory.

By the Way

> To add to the confusion, most Unix people don't pronounce the slashes, particularly if the first component of the filename is a well-known directory. I would pronounce `/home/taylor` as "home taylor," but I would usually pronounce `/newt/awk/test` as "slash newt awk test." When in doubt, pronounce the slash.

As you learn more about Unix, particularly about how to navigate in the file system, you'll find that a clear understanding of the difference between a relative and absolute filename proves invaluable. The rule of thumb is that if a filename begins with `/`, it's absolute.

Task 3.2: Hidden Files in Unix

One of the best aspects of living in an area for a long time, frequenting the same shops and visiting the same restaurants, is that the people who work at each place learn your name and preferences. Many Unix applications can perform the same trick, remembering your preferred style of interaction, what files you last worked with, which lines you've edited, and more, through *preference files*.

On the Macintosh, there's a folder within each user's home directory called Library. Within that there's another folder called Preferences, which is a central storage place for preference files, organized by application. On my Macintosh, for example, I have about 75 different preference files in this directory, enabling me to have all my programs remember the defaults I prefer.

Unix must support many users at once, so Unix preference files can't be stored in a central spot in the file system. Otherwise, how would the system distinguish between your preferences and those of your colleagues? To avoid this problem, all Unix applications store their preference files in your home directory.

Programs want to be able to keep their own internal preferences and status stored in your directory, but these aren't for you to work with or alter. If you use DOS, you're probably familiar with how Windows solves this problem: Certain files are hidden and do not show up when you use `DIR`, in DOS, or the File Manager to list files in a directory.

Macintosh people don't realize it, but the Macintosh also has lots of hidden files. On the topmost level of the Macintosh file system, for example, the following files are present, albeit hidden from normal display: .DS_Store, Desktop DB, .VolumeIcon.icns, NavMac8000QSFile, and my personal favorite .symSchedScanLockxz. Displaying hidden files on the Macintosh is very difficult, as it is with Windows.

Fortunately, the Unix rule for hiding files is much easier than that for either the Mac or PC. No secret status flag reminds the system not to display the file when listing directories. Instead, the rule is simple, any filename starting with a dot (`.`) is called a *hidden file*.

By the Way

A *hidden file* is any file with a dot as the first character of the filename.

If the filename or directory name begins with a dot, it won't show up in normal listings of that directory. If the filename or directory name has any other character as the first character of the name, it lists normally.

1. Knowing that, turn to your computer and enter the `ls` command to list all the files and directories in your home directory.

```
% ls -F
Archives/       Mail/           RUMORS.18Sept   mailing.lists
InfoWorld/      News/           bin/            newlists
LISTS           OWL/            iecc.list       src/
%
```

2. You can see that I have 12 items in my own directory, 7 directories (the directory names have a slash as the last character because of the `-F`, remember), and 5 files. Files have minimal rules for naming, too. Avoid slashes, spaces, and tabs, and you'll be fine.

3. Without an explicit direction to the contrary, Unix is going to let the hidden files remain hidden. To add the hidden files to the listing, you just need to add a `-a` flag to the command. Turn to your computer and try this command to see what hidden files are present in your directory. These are my results:

```
% ls -aF
./              .gopherrc       .oldnewsrc      .sig            RUMORS.18Sep
../             .history*       .plan           Archives/       bin/
.Agenda         .info           .pnewsexpert    InfoWorld/      iecc.list
.aconfigrc      .letter         .report         LISTS           mail.lists
.article        .login          .rm-timestamp   Mail/           newlists
.cshrc          .mailrc         .rnlast         News/           src/
.elm/           .newsrc         .rnsoft         OWL/
%
```

Many dot files tend to follow the format of a dot, followed by the name of the program that owns the file, with `rc` as the suffix. In my directory, you can see six dot files that follow this convention: `.aconfigrc`, `.cshrc`, `.gopherrc`, `.mailrc`, `.newsrc`, and `.oldnewsrc`.

Because of the particular rules of hidden files in Unix, they are often called dot files, and you can see that I have 23 dot files and directories in my directory.

By the Way

The rc suffix tells you that this file is a configuration file for that particular utility. For instance, .cshrc is the configuration file for the C shell and is executed every time the C shell (/bin/csh) is executed. You can define aliases for C shell commands and a special search path, for example.

Because it's important to convey the specific filename of a dot file, pronunciation is a little different from elsewhere in Unix. The name `.lynxrc` would be spoken as "dot lynx are sea," and `.mailrc` would be "dot mail are sea." If you can't pronounce the program name, odds are good that no one else can either, so `.cshrc` is "dot sea ess aitch are sea."

Other programs create many different dot files and try to retain a consistent naming scheme. You can see that `.rnlast` and `.rnsoft` are both from the `rn` program, but it's difficult to know simply from the filenames that `.article`, `.letter`, `.newsrc`, `.oldnewsrc`, and `.pnewsexpert` are all also referenced by the `rn` program. Recognizing this problem, some application authors designed their applications to create a dot directory, with all preference files neatly tucked into that one spot. The `elm` program does that with its `.elm` hidden directory.

Some files are directly named after the programs that use them: the `.Agenda` file is used by the `agenda` program, and `.info` is used by the `info` program. Those almost have a rule of their own, but it's impossible to distinguish them from `.login`, from the `sh` program; `.plan` for the `finger` program; `.rm-timestamp` from a custom program of my own; and I frankly have no idea what program created the `.report` file!

This should give you an idea of the various ways that Unix programs name and use hidden files. As an exercise, list all the dot files in your home directory and try to figure out which program created each file. Check by looking in the index of this book to see whether a program by that name exists, if it's a ".xxx" file. If you can't figure out which programs created which files, you're not alone. Keep the list handy; refer to it as you learn more about Unix while exploring *Sams Teach Yourself Unix in 24 Hours, Fourth Edition*, and by the time you're done, you'll know exactly how to find out which programs created which dot files.

Task 3.3: The Special Directories . and ..

I haven't mentioned two dot directories, although they show up in my listing and most certainly show up in your listing too. They are dot and dot dot (`.` and `..`), and they're shorthand directory names that can be terrifically convenient.

The *dot* directory is shorthand for the current location in the directory hierarchy; the *dot-dot* directory moves you up one level, to the parent directory.

Consider again the list of files shown in Figure 3.3. If you were looking at the files in the `California Personnel` directory (best specified as `/California/Personnel`) and wanted to check quickly an entry in the `Bldgs` file for `California`, either you'd have to use the absolute filename and enter the lengthy `ls /California/Bldgs`, or, with the new shorthand directories, you could enter `ls ../Bldgs`.

As directories move ever deeper into the directory hierarchy, the dot-dot notation can save you much typing time. For example, what if the different states and related files were all located in my home directory `/home/taylor`, in a new directory called `business`? In that case, the absolute filename for employee `Raab,M.` in California would be `/home/taylor/business/California/Personnel/Raab,M.`, which is unwieldy and a great deal to type if you want to hop up one level and check on the `buildings` database in `Indiana`!

You can use more than one dot-dot notation in a filename too, so if you're looking at the `Raab,M.` file and want to check on `Dunlap,L.`, you could save typing in the full filename by instead using `../../../Washington/Personnel/Dunlap,L.`. Look at Figure 3.3 to see how that would work, tracing back one level for each dot-dot in the filename.

This explains why the dot-dot shorthand is helpful, but what about the single-dot notation that simply specifies the current directory?

I haven't stated it explicitly yet, but you've probably figured out that one ramification of the Unix file system organization, with its capability to place applications anywhere in the file system, is that the system needs some way to know where to look for particular applications. Just as if you were looking for something in a public library, in Unix, having an understanding of its organization and a strategy for searching is imperative for success and speed.

Unix uses an ordered list of directories called a *search path* for this purpose. The search path typically lists five or six different directories on the system where the computer checks for any application you request.

The question that arises is, "What happens if your own personal copy of an application has the same name as a standard system application?" The answer is that the system always finds the standard application first, if its directory is listed earlier in the search path.

To avoid this pitfall, use the dot notation, forcing the system to look in the current directory rather than search for the application. If you wanted your own version of the `ls` command, for example, you'd need to be in the same directory as the command and enter `./ls` to ensure that Unix uses your version rather than the standard version.

1. Enter `./ls` on your computer and watch what happens.
2. Enter `ls` without the dot notation, and you'll instantly see how the computer searches through various directories in the search path, finds the `ls` program, and executes it, automatically.

When you learn about `cd` (change directory) later in this hour, you also will learn other uses of the dot-dot directory, but the greatest value of the dot directory is that you can use it to force the system to look in the current directory and nowhere else for any file specified.

Task 3.4: The `env` Command

You've learned much about the foundations of the Unix file system and how applications remember your preferences through hidden dot files. There's another way, however, that the system remembers specifics about you, and that's through your *user environment*. The user environment is a collection of specially named variables, mnemonically named values, that have specific values.

1. To view your environment, you can use the `env` command. Here's what I see when I enter the `env` command on my system:

```
% env
HOME=/home/taylor
SHELL=/bin/csh
TERM=vt100
PATH=/home/taylor/bin:/bin:/usr/bin:/usr/ucb:/usr/local/bin:
➥/usr/unsup/bin:.
MAIL=/usr/spool/mail/taylor
LOGNAME=taylor
TZ=EST5
%
```

Try it yourself and compare your values with mine. You might find that you have more defined in your environment than I do because your Unix system uses your environment to keep track of more information.

Here we see some of the standard environment variables. Table 3.1 describes what they do.

TABLE 3.1 Common Shell Environment Variables and What They Do

Variable	Description
HOME	The directory where you log in and store all your personal files
SHELL	The program you run as your command-line interpreter
TERM	The type of terminal emulation you need to provide cursor graphics
PATH	A list of directories searched when you enter a command
MAIL	The file where your incoming mail is stored
LOGNAME	Your login name
TZ	The time zone on your system

By the Way

Many Unix systems offer the `printenv` command instead of `env`. If you enter `env` and the system complains that it can't find the `env` command, try using `printenv` instead. All examples here work with either `env` or `printenv`.

Task 3.5: PATH and HOME

The two most important values in your environment are the name of your home directory (`HOME`) and your search path (`PATH`). Your home directory (as it's known) is the name of the directory that you always begin your Unix session within.

The `PATH` environment variable lists the set of directories, in left-to-right order, that the system searches to find commands and applications you request. You can see from the example that my search path tells the computer to start looking in the `/home/taylor/bin` directory, and then sequentially try `/bin`, `/usr/bin`, `/usr/ucb`, `/usr/local/bin`, `/usr/unsup/bin`, and `.` before concluding that it can't find the requested command. Without a `PATH`, the shell wouldn't be able to find any of the many, many Unix commands: As a minimum, you always should have `/bin` and `/usr/bin`.

1. You can use the `echo` command to list specific environment variables too. Enter `echo $PATH` and `echo $HOME`. If you forget the "$" then the shell doesn't know you are specifying that you want to know the value of the named variable. Try it, you'll see what I mean.

   When I enter these two `echo` statements as shown above, I get the following results:

```
% echo $PATH
/home/taylor/bin:/bin:/usr/bin:/usr/ucb:/usr/local/bin:/usr/unsup/
➥bin:.
% echo $HOME
/home/taylor
%
```

   Your `PATH` value is probably similar, although certainly not identical, to mine, and your `HOME` is `/home/`*`accountname`* or similar (*`accountname`* is your account name).

Task 3.6: Find Where You Are with `pwd`

So far you've learned a lot about how the file system works but not much about how to move around in the file system. With any trip, the first and most important step is to find out your current location—that is, the directory in which you are currently working. In Unix, the command `pwd` tells you the *present working directory*.

1. Enter `pwd`. The output should be identical to the output you saw when you entered `env HOME` because you're still in your home directory.

```
% echo $HOME
/home/taylor
% pwd
/home/taylor
%
```

Think of `pwd` as a compass, always capable of telling you where you are. It also tells you the names of all directories above you because it always lists your current location as an absolute directory name.

Task 3.7: Moving to Another Location with `cd`

The other half of the dynamic duo is the `cd` command, which is used to change directories. The format of this command is simple, too: `cd` *`new-directory`* (where *`new-directory`* is the name of the new directory you want).

1. Try moving to the very top level of the file system and entering `pwd` to see whether the computer agrees that you've moved.

```
% cd /
% pwd
/
%
```

2. Notice that `cd` doesn't produce any output. Many Unix commands operate silently like this, unless an error is encountered. The system then indicates the problem. You can see what an error looks like by trying to change your location to a nonexistent directory. Try the `/taylor` directory to see what happens!

```
% cd /taylor
/taylor: No such file or directory
%
```

3. Enter `cd` without specifying a directory. What happens? I get the following result:

```
% cd
% pwd
/home/taylor
%
```

4. Here's where the `HOME` environment variable comes into play. Without any directory specified, `cd` moves you back to your home directory automatically. If you get lost, it's a fast shorthand way to move to a known location without fuss.

   Remember the dot-dot notation for moving up a level in the directory hierarchy? Here's where it also proves exceptionally useful. Use the `cd` command without any arguments to move to your home directory, and then use `pwd` to ensure that's where you've ended up.

5. Now, move up one level by using `cd ..` and check the results with `pwd`:

```
% cd
% pwd
/home/taylor
% cd ..
% pwd
/home
%
```

By the Way

Use the `ls -C -F` command to list all the directories contained at this point in the file system. Beware, though; on large systems, this directory could easily have hundreds of different directories. On one system I use, almost 550 different directories are on one level above my home directory in the file system!

```
% ls -F
armstrong/  christine/  guest/   laura/   matthewm/ shane/
bruce/      david/      higgins/ mac/     rank/     taylor/
cedric/     green/      kane/    mark/    shalini/  vicki/
%
```

Try using a combination of `cd` and `pwd` to move about your file system, and remember that without any arguments, `cd` always zips you right back to your home directory.

Summary

This hour focused on the Unix hierarchical file system. You've learned the organization of a hierarchical file system, how Unix differs from Macintosh and DOS systems, and how Unix remembers preferences with its hidden dot files. This hour also explained the difference between relative and absolute filenames, and you've learned about the `.` and `..` directories. You learned four new commands: `env` to list your current environment, `echo` to show a particular value, `cd` to change directories, and `pwd` to find your present working directory location.

Workshop

The Workshop summarizes the key terms you learned and poses some questions about the topics presented in this chapter. It also provides you with a preview of what you will learn in the next hour.

Key Terms

absolute filename Any filename that begins with a leading slash (/); these always uniquely describe a single file in the file system.

binary A file format that is intended for the computer to work with directly rather than for humans to peruse. See also **executable**.

device driver All peripherals attached to the computer are called *devices* in Unix, and each has a control program always associated with it, called a *device driver*. Examples are the device drivers for the display, keyboard, mouse, and all hard disks.

directory A type of Unix file used to group other files. Files and directories can be placed inside other directories, to build a hierarchical system.

directory separator character On a hierarchical file system, there must be some way to specify which parts of a full filename are directories and which part is the actual filename itself. This becomes particularly true when you're working with

absolute filenames. In Unix, the directory separator character is the slash (/), so a filename like `/tmp/testme` is easily interpreted as a file called `testme` in a directory called `tmp`.

dot A shorthand notation for the current directory.

dot dot A shorthand notation for the directory one level higher up in the hierarchical file system from the current location.

dot file A configuration file used by one or more programs. These files are called dot files because the first letter of the filename is a dot, as in `.profile` or `.login`. Because they're dot files, the `ls` command doesn't list them by default, making them also hidden files in Unix. See also **hidden file**.

executable A file that has been set up so that Unix can run it as a program. This is also shorthand for a binary file. You also sometimes see the phrase *binary executable*, which is the same thing. See also **binary**.

hidden file By default, the Unix file-listing command `ls` shows only files whose first letter isn't a dot (that is, those files that aren't dot files). All dot files, therefore, are hidden files, and you can safely ignore them without any problems. Later, you learn how to view these hidden files. See also **dot file**.

home directory This is your private directory and is also where you start out when you log in to the system.

kernel The underlying core of the Unix operating system itself. This is akin to the concrete foundation under a modern skyscraper.

preference file These are what dot files (hidden files) really are: They contain your individual preferences for many of the Unix commands you use.

relative filename Any filename that does not begin with a slash (/) is a filename whose exact meaning depends on where you are in the file system. For example, the file `test` might exist in both your home directory and in the root directory: `/test` is an absolute filename and leaves no question which version is being used, but `test` could refer to either copy, depending on your current directory.

root directory The directory at the very top of the file system hierarchy, also known as *slash*.

search path A list of directories used to find a command. When a user enters a command `ls`, the shell looks in each directory in the search path to find a file `ls`, either until it is found or the list is exhausted.

slash The root directory.

symbolic link A file that contains a pointer to another file rather than contents of its own. This can also be a directory that points to another directory rather than having files of its own. A useful way to have multiple names for a single program or to allow multiple people to share a single copy of a file.

user environment A set of values that describe the user's current location and modify the behavior of commands.

working directory The directory where the user is working.

Exercises

1. Can you think of information you work with daily that's organized in a hierarchical fashion? Is a public library organized hierarchically?

2. Which of the following files are hidden files and directories according to Unix?

```
.test   hide-me   ,test   .cshrc
../     .dot.     dot     .HiMom
```

3. What programs most likely created the following dot files and dot directories?

```
.cshrc    .rnsoft     .exrc      .print
.tmp334   .excel/     .letter    .vi-expert
```

4. In the following list, identify the items that are absolute filenames:

```
/Personnel/Taylor,D.
/home/taylor/business/California
../..
Recipe:Gazpacho
```

5. Referring to the list of directories found on all Unix systems (`/bin`, `/dev`, `/etc`, `/lib`, `/lost+found`, `/mnt`, `/sys`, `/tmp`, `/usr`), use `cd` and `pwd` to double-check that they are all present on your own Unix machine.

Preview of the Next Hour

In the next hour, you learn all about the `ls` command that you've been using, including an extensive discussion of command flags. The command `touch` enables you to create your own files, and `du` and `df` help you learn how much disk space is used and how much is available, respectively. You also learn how to use a valuable if somewhat esoteric Unix command, `compress`, which helps you minimize your disk-space usage.