UnixWorld ‘‘New To Unix’’ Column: August 1988

Learning Essential Command and Files

By Augie Hansen

This month we will learn about commands, which are the tools you use to get a UNIX system to do you bidding. You interact with UNIX by typing commands. Your commands are read and interpreted by a user-interface program called a shell.

The UNIX shell tells you that it is waiting for a command by printing a prompt. The AT&T version of UNIX running the standard Bourne shell or the newer Korn shell uses a dollar sign (`$`) as the default prompt. Berkeley UNIX uses the C shell, which prints a percent signal (`%`) as the prompt. Your actual prompt may differ from the default if your system administrator has customized it. Although the various shells differ in the details, they are virtually identical in purpose and are similar in their basic operation.

When you press a key on your terminal, a numeric code that represents that character is generated and sent to the host computer. When a number is sent to your terminal, the character that it represents is displayed. Several codes stand for control characters that have an action, such as moving to the beginning of the current line (carriage return), but no graphic representation.

In general, when you type a character at the keyboard, a copy is echoed immediately back to you so you can see what you just typed. The other copy, which is destined for a UNIX program, is buffered into a line. A buffer is an electronic equivalent of a reservoir. It collects characters you type, but doesn’t release them until told to do so. The Return key is the command to pass the input buffer’s contents to the shell or other running program waiting for your input. This is why you have to type Return after you type your log-in name and password when you log in. Note that the system echoed back your log-in name as you typed it, but not your password.

Some Simple Commands

Here are a few commands that produce output without requiring much input from you. If you make a typing mistake when entering commands, you can use Backspace key to back up and correct the error. The `date` command prints the current date and time. Listing 1A shows the command and the form of the output it produces.

If you want to know what day of the week Christmas will fall on this year, use the `cal` command to print out a calendar for December. The `cal` command without any arguments (words on the command line following the command name itself) prints a calendar for the current month. Listing 1B shows the command with two arguments: the ‘‘12’’ specifies the desired month (1-12), and the ‘‘1988’’ specifies the year. If your provide only a single argument, it must be the year, and the year must be specified in long form. If you type 88 for the year argument, it literally means 88 A.D., not 1988.

If you want to know about the hardware and software system you are working on, perhaps to check on software compatibility for some new program you have obtained, use the `uname` command. Listing 1C shows the output produced by `uname` with the `-a` option set. The output shows that UNIX System V is running on a system called omni. The software release is 3.5.1, and the the hardware is based on a Motorola 68000 microprocessor. This system is an AT&T 3B1, one of several UNIX and UNIX-like systems I use in my work.

UNIX commands, for the most part, have been designed to operate reasonably when issued without optional arguments, but most let you change their default behavior by typing *options*. Always type a space after the command name and between each element (word) that you type on the command line.

An option consists of an option flag (usually a dash) followed by an option character or word. The `uname` command, for example, accepts options to print out the node name (`-n`), which is omni in this example, the operating system version (`-v`), and various other information about the host system. The `-a` option requests a printout of all system information.

Some command options require additional information following the option character or word, as we'll see later.

File and Directory Commands

UNIX maintains information as a collections *files* and *directories*. So what are files and directories?

A file is a collection of characters, and each file has a name by which you access it. The names of files are kept in directories. A directory serves the same purpose as the table of contents of a book. An entry in a directory points to the information related to file name, which gives you ready access to the file's contents while relieving you of any concern about where the information is actually stored.

A directory is really just a file, but one that has a specific form for each of its entries. This design results in what is described as a directory hierarchy in which each each directory can contain the names of ordinary files and the names of other directories.

Figure 1 shows a sample directory hierarchy. Each boxed item is the name of a directory, and each unboxed item is a file. The highest level of the UNIX directory hierarchy is called *root*, which is symbolized by the forward slash (`/`) character.

Extending downward from the root is a set of system directories. On each UNIX system, at least one of those directories is allocated to users. On an AT&T 3B1, the user file space, as it is called, is usually `/u`, where `/` is root and `u` is the directory name.

Different Naming Conventions

Other systems may use different naming conventions. But when you log in, you will be automatically placed in the correct directory, which is called your *home* directory. In my case, that's `/u/arh`. Here, `/u/arh` is called a *full path name* because it starts at root and extends to the directory or file name. For instance, the full path name of the file that hold this month's column is `/u/arh/uworld/newuser/08cmds.sa` on my system.

Here are a few of the simple commands that deal with files and directories. You use the `ls` command to list the contents of a directory. Simply typing ‘‘ls’’ requests a listing of the current directory, as shown in Listing 2A. If you provide a directory name as an argument, you are given a listing of the named directory. Using the `-l` option with `ls` produces a long listing (Listing 2B), which contains considerable detail about each file and directory entry, including ownership, access permissions, and file date and time stamps.

If you don’t know the name of the directory you are working in, type ‘‘pwd’’ to find out. The command prints the name of the working directory, which is usual called the *current* directory.

Use the `cd`, or change directory command to move about in the directory hierarchy. To change to a directory immediately subordinate to the current directory, simply type `cd` and the directory name as an argument. To change to the directory immediately above the current directory, type `cd ..`, or type the full path name of the destination directory. The special name `..` (called ‘‘dot dot’’) is a shorthand notation for the parent directory, which is the directory above the current directory. Each directory in the hierarchy except root has a parent directory (because root is its own parent directory).

Elementary Editing

You’ll spend much of your time creating and editing files. The primary tool you use for this work is a text editor, or possibly a word processor. To get you started, we will look at the standard UNIX line-oriented editor, Ed. The Ed editor was designed to be small and fast. It was not designed with ease of use in mind, at least not when judged by today’s standards.

Listing 3 shows the major elements of a dialog between the user and Ed during the creation of a file called `sample.txt`. Because the file does not exist, Ed prints a message (`?sample.txt`) to let you know it couldn’t find a file by that name. To create the file, use the append (`a`) command to put the editor into the text mode. (Note that all Ed commands are followed by a Return.) Then type your input a line at a time. If you make a mistake, use the Backspace key to move back over the error and retype.

To switch back to command mode, type a ‘‘.’’ (dot) on a line by itself and press Return.

The Ed program worked in a temporary editing buffer as you input and edit text. You must write the buffer to a disk file to maintain a nonvolatile copy of it. Once back in the command mode, you can save the file on disk by giving the write (`w`) command. The editor responds by telling you how many bytes were saved.

To view what you have written, use the print (`p`) command. You can provide a range of lines to the print command, and in Listing 3B, the range is `1,$`. This notation means print all lines between line one and the last line of the file (denoted by $) inclusive. The term *print* is literal in the case of a hard-copy terminal. But on a video terminal, a more meaningful term is *display*.

Next month we will continue exploring the UNIX editors by looking more closely at the Ed editor and by introducing Vi, the UNIX full-screen visual editor.

Edited by Becca Thomas / Online Editor / UnixWorld Online / editor@unixworld.com

Go To Content
Search

Text entry and HTML markup by Arlene Lee
Last Modified: Monday, 18-Mar-96 07:03:52 PST