

CHAPTER 7

Introduction to the C Shell

Different Shells

Most UNIX variants allow you to select among several shells. The shell is important to you as a user, because it is your window into the system. The many shells available on UNIX variants are similar in that you use them to issue commands, control many aspects of your user environment, write command files and shell programs, and so on. Because you'll probably be spending a lot of time in your shell you should develop an understanding of several different shells and see if you develop a preference for one over the other. A particular shell, such as Bash, does not vary much from one UNIX variant to another. The shells themselves, however, have some unique features. In general, I don't find that users strongly prefer one shell over another. All the shells are similar in functionality and enjoyable to use once you get to know them. Most UNIX variants allow your system administrator to select from among several different shells when configuring users, so there is usually some flexibility concerning the shell that users run. In general, system administrators prefer users to use the same shell, making system administration easier in general. Most sys-

tem administrators, however, are happy to grant a user request for a particular shell if indeed it is available on your system and you have a good reason you'd like to use it. I'll cover the C shell in this chapter. Bash and KornShell were covered in the two previous chapters.

Introduction to the C Shell

The C shell is similar to other shells in that it provides a user interface to UNIX. You can use the C shell in the following three ways:

- Interactively type commands on the command line.
- Group commonly executed sets of commands into command files that you can execute by typing the name of the file.
- Create C shell programs using the structured programming techniques of the C shell.

These three techniques are listed in the order in which you'll probably use them. First, you log in and use interactive commands. Then you group together commonly used commands and execute them with a single command. Finally, you may want to create sophisticated shell scripts.

In this chapter I cover login and interactive commands and a lot of useful ways to use the C shell.

Most of the examples in this chapter are from Solaris and HP-UX systems. You probably will find both your user setup and the operation of the C shell on other systems similar to what is covered in this chapter. Much of the setup of any shell is performed by the system administrator, so you will surely find differences in your C shell setup compared with what is shown in this chapter. In general, however, the operation of the C shell is similar from one system to another.

Issuing Commands

The first activity you perform after you log into the system is to issue commands at the prompt. A command you may want to issue immediately is **ls -al**. Here is what I see on my system after executing this:



```
sys1 7: ls -al
total 10
drwxr-x---  2 marty2  users      96 May  5 09:34 .
drwxr-xr-x 10 root    root      1024 May  5 10:38 ..
-rw-r--r--  1 marty2  users      814 May  5 09:34 .cshrc
-rw-r--r--  1 marty2  users      347 May  5 09:34 .exerc
-rw-r--r--  1 marty2  users      341 May  5 09:34 .login
-rw-r--r--  1 marty2  users      446 May  5 09:34 .profile
sys1 8:
```

The C shell prompt consists of system name (sys1) followed by the command number and a colon. I cover the prompt shortly.

ls -al shows two files related to the C shell in this user area:

.cshrc and **.login**

Figure 7-1 is the contents of **.cshrc**.

```
# Default user .cshrc file (/usr/bin/csh initialization).

# Usage: Copy this file to a user's home directory and edit it to
# customize it to taste. It is run by csh each time it starts up.

# Set up default command search path:
#
# (For security, this default is a minimal set.)

    set path=( $path )

# Set up C shell environment:

if ( $?prompt ) then      # shell is interactive.
    set history=20        # previous commands to remember.
    set savehist=20      # number to save across sessions.
    set system=`hostname` # name of this system.
    set prompt = "$system \!: " # command prompt.

# Sample alias:

alias    status `(date; bdf)`

# More sample aliases:

alias  d    dirs
alias  pd   pushd
alias  pd2  pushd +2
alias  po   popd
alias  m    more
endif
```

Figure 7-1 Sample **.cshrc**

Figure 7-2 shows the contents of **.login**.

```
# @(#) $Revision: 72.3 $

# Default user .login file ( /usr/bin/csh initialization )

# Set up the default search paths:
set path=( $path )

#set up the terminal
eval `tset -s -Q -m '?:hp' `
stty erase "^H" kill "^U" intr "^C" eof "^D" susp "^Z" hupcl ixon ixoff
tostop
tabs

# Set up shell environment:
set noclobber
set history=20
```

Figure 7-2 Sample **.login**

The **.cshrc** File

The sequence of events after login varies from one UNIX system to another. On many systems, the **.cshrc** is first read and executed by the C shell. You can modify the **.cshrc** file to specify the command-line prompt you wish to use, initialize the history list, and define aliases. The upcoming sections describe the way the **.cshrc** file shown in Figure 7-1 defines these. Let's first take a quick look at the **.login** file in the next section.

The .login File

On many UNIX systems the **.login** file is read after the **.cshrc** file. There are only two issues related to setup present in the example shown. The first is the **tset** command, which sets the **TERM** environment variable. The **eval** preceding **tset** means that the C shell executes **tset** and its arguments without creating a child process. This allows **tset** to set environment variables in the current shell instead of a subshell, which would be useless. The **stty** command is used to set terminal I/O options. The two set commands are used to define shell variables, which I describe shortly. The **noclobber** does not permit redirection to write over an existing file. If you try to write over an existing file, such as **/tmp/processes** below, you receive a message that the file exists:



```
sys1 1: ps -ef > /tmp/processes
/tmp/processes: File exists
```

The ">" means to take the output of **ps** and rather than write it to your screen, write it to **/tmp/processes**. The file **/tmp/processes** will not be written over, however, with the output of **ps -ef** because **/tmp/processes** already exists and an environment variable called **noclobber** has been set. If **noclobber** is set, then redirecting output to this file will not take place. This is a useful technique for preventing existing files from being accidentally overwritten. There are many forms of redirection that you'll find useful. Redirection is covered later in this chapter.

Initialize History List in `.cshrc`

The C shell can keep a history list of the commands you have issued. If you wish to reissue a command or view a command you earlier issued, you can use the history list.

The commands issued are referred to by number, so you want to have a number appear at the command prompt. The following line in `.cshrc` provides a number following the system name:

```
set prompt = "$system \!:"  
sys1 1:
```

We get into shell and environment variables shortly, but for now it is sufficient to know that `$system` corresponds to system name "sys1."

You can specify the number of previously issued commands you want to save and view when you issue the `history` command. The following line in `.cshrc` sets the history list to 20:

```
set history = 20
```

The last 20 commands issued are displayed when you issue the `history` command.

The `savehist` variable allows you to save a specified number of history commands after logout. By default, when you log out, the history list is cleared. This variable has a value of 20, so that upon the next login there will be 20 commands from the previous session will be saved.

Command-Line History

You can view the history list a variety of different ways. Let's first view the last 20 commands by simply issuing the **history** command:

```
sys1 23: history
      4 whoami
      5 pwd
      6 find / -name login -print &
      7 hostname
      8 who
      9 more /etc/passwd
     10 history
     11 history 5
     12 echo $PATH
     13 more .login
     14 cat .login
     15 exit
     16 exit
     17 history -h
     18 pwd
     19 whoami
     20 cd /tmp
     21 cat database.log
     22 cd
     23 history
sys1 24:
```

We can also print the history list without line number, as shown in the following example:

```
sys1 24: history -h
pwd
find / -name login -print &
hostname
who
more /etc/passwd
history
history 5
echo $PATH
more .login
cat .login
exit
```

```
exit
history -h
pwd
whoami
cd /tmp
cat database.log
cd
history
history -h
sys1 25:
```

Next let's print the history list in reverse order:

```
sys1 25: history -r
25 history -r
24 history -h
23 history
22 cd
21 cat database.log
20 cd /tmp
19 whoami
18 pwd
17 history -h
16 exit
15 exit
14 cat .login
13 more .login
12 echo $PATH
11 history 5
10 history
9 more /etc/passwd
8 who
7 hostname
6 find / -name login -print &
sys1 26:
```

We can also select the number of events we want to print from the history list. The following example prints only the last ten commands from the history list:

```
sys1 26: history 10
17 history -h
18 pwd
```