Unix—the Bare Minimum

Norman Matloff

September 27, 2005

Contents

1 Purpose

The information here is intended to be a review for those who have had a bit of prior exposure to Unix, and as a quick introduction to Unix for those who have never seen it before. (Some of the material may be new even to those with some prior exposure to Unix.)

2 Shells

A **shell** is a program[1] that inputs Unix commands from the keyboard and relays them to the Unix system for execution. Shells typically include various shortcuts for users to use in stating their commands, and also a programming feature, in which users can make programs out of sets of their commands.

The first popular Unix shell was the Bourne shell, named **sh**. It still very popular, in a modernized version called the Bourne Again Shell, **bash**.

There are many other shells. Our UCD CSIF system accounts are set up so that your login shell is the C-shell; its official name, taken as a command itself, is **csh**. We actually use an extension of **csh**, called **tcsh**.

2.1 Switching from One Shell to Another

If you wish to temporarily use another shell, just type its name in an existing shell, e.g.

```
$ bash
```

That will start an instance of the **bash** program (executing within your original C-shell, but that won't matter). Or if someone has given you a **script**, i.e. a file **x.sh** containing **bash** commands, type

```
$ bash x.sh
```

If you wish to make **bash** your default shell, use the **chsh** ("change shell") command:

```
$ chsh -s bash
```

This introduction will focus mainly on the C-shell, but once you learn the material here, it will be easy to learn other shells, say BASH. There are many tutorials on the latter on the Web, e.g. at `http://pegasus.rutgers.edu/~elflord/unix/bash-tute.html`.

2.2 Shell Conveniences

A nice feature of modern shells is **command line editing**. Make sure to use it! Here is how it works in the C and BASH shells:

Say I wish to type a command

[1] Yep, it's a program, likely written in C. You could write a shell too, with a little knowledge of Unix processes.

```
cd /altpkg/tex/texmf/fonts/tfm/public/cm
```

(As you will see later, the **cd** command changes directories, but don't think about that now.) Suppose, though, that I mistype it as

```
cd /altpgk/tex/texmf/fonts/tfm/public/cm
```

Suppose I have not yet hit the return key. Then I can go back to change the "gk" as follows: use the left-arrow key to go to the 'k'; hit the Delete key, which will remove the 'g'; use the right-arrow key to go to the '/'; hit the g key to put the 'g' back in; and then hit the return key to process the command.

On the other hand, if I have already hit the return key when I notice my typing error, I can use ctrl-p to go back to my previous command. (Use ctrl-p to go back through several previous commands, and ctrl-n to go forward.) Note that you can also use this to repeat (without change) a previous command.

Once you get used to this, it saves you a lot of typing, and allows you to concentrate better on your work.

The **tcsh** (and some other shells, such as **bash**) also allows you to do **file name completion**, again a great saver of typing and time.

Suppose for example I have a file named **jack.and.the.beanstock**, which I need to copy to a file name **gy**. The command, which you will learn later in this tutorial, is

```
cp jack.and.the.beanstock gy
```

But rather than typing that long name by hand, suppose that this is the only file which begins with "ja". What I can do is type

```
cp ja
```

and then hit the Tab key. The shell will then complete that file name for me, so that the command line on the screen will now be

```
cp jack.and.the.beanstock
```

I now continue typing, adding " gy", producing

```
cp jack.and.the.beanstock gy
```

and hit the return key.

Many text editors, e-mail utility programs and so on include some kind of file-completion feature.

Note carefully that **tcsh** is an extension of **csh**. The **tcsh** version does use a different startup file, **/.tcshrc**,[2] but will use **csh**'s counterpart, **/.cshrc**, if **/.tcshrc** is not there. I recommend that you NOT have a **/.tcshrc** file (remove it if your system has already placed one there), and that you use **/.cshrc** instead; that way you get the same environment when you run either **csh** or **tcsh**. (Often free software downloaded from the Web will use **csh**.) The startup file for **bash** is **/.bashrc**.

[2]The " /" means your home directory.

3 Files and Directories

Unix uses a **hierarchical** file system, meaning the following. When you first log in, you will be at a point in your file system known as your **home directory**. Within that directory you can make subdirectories, and within them you can make sub-subdirectories, and so on. So, your file system has a tree-like shape. (And your file system is in turn a subtree of the collection of all files on the machine.)

This hierarchical system helps you to organize your files. For example, suppose you are taking ECS 40 and Stat 32. You could make directories with these names, and then keep all your files for a given class in that directory. Similarly, you may be looking for a job, so you might create a directory named JobHunting, and then keep all your resume's, cover letters and so on in that directory.

3.1 Creating Directories

To create a directory, use the **mkdir** command. Suppose, for example, you are in your home directory, and wish to make a subdirectory named ECS40, as suggested above. You could type

```
mkdir ECS40
```

3.2 Moving to Other Directories

To go from one directory to another, use the **cd** command. For example, if you are currently in your home directory and you have created the ECS40 subdirectory, simply type

```
cd ECS40
```

However, suppose you are currently in the Stat32 subdirectory, and you wish to go to ECS40. The above command won't work, since ECS40 is not a subdirectory of Stat32. Instead, you can type

```
cd ~/ECS40
```

with the tilde mark ŝignifying your home directory. In other words, you are saying, "Change to ECS40, which is a subdirectory of my home directory."

The directory which is up one level in the directory tree can be referred to as "..". Thus for example,

```
cd ..
```

would take you up to that level. As another example, an alternate method for moving from the Stat32 directory to the ECS40 directory in the example above would be

```
cd ../ECS40
```

3.3 Some Directory and File Commands

3.4 pwd Command: Which Directory Are We In?

To see which directory you are currently in (theoretically you should know, but sometimes you might lose track of which directory you are in), type the **pwd** command.

3.5 ls Command: What Files Are Here?

To get a list of all files you have in the current directory, use the **ls** command. This command is more useful, though, if you use the -F option, i.e. you type

```
ls -F
```

This will tell you which files are executable (their names will be appended with asterisks), and which are subdirectories (their names will be appended with slashes). By the way, **ls** will not report files whose names begin with a period (there typically are several of these); to see those, add the -a option, e.g. type 'ls -Fa'.

3.6 rm Command: How to Remove Files

To remove a file, use the **rm** command. E.g.

```
rm x
```

will result in the file x being deleted.

3.7 cp Command: Copying Files

Say you have a file x and wish to make a new copy of it in a file named y. Simply type

```
cp x y
```

Say you have files u and v and wish to copy them to files named u and v within a directory w. Type

```
cp u v w
```

See the man page (see "Online Help" later in this tutorial) for many more things you can do with cp.

3.8 mv Command: Renaming Files

To rename a file, use **mv**. E.g.

```
mv x y
```

means "move x to y," i.e. take the file x and rename it as y.

3.9 Applying Commands to Other Directories

All these commands above can be used on directories other than the current one. For example, if we are currently in the directory Stat32 but wish to know what files are in the directory ECS40, we could use **cd** to go to the latter directory and then use **ls** there, but it is easier to just type

```
ls ~/ECS40
```

from Stat32, not leaving that directory.

Similarly, if we had a file z in Stat32 which we wanted to copy to ECS40, we could type (from the Stat32 directory)

```
cp z ~/ECS40/z
```

In fact, even

```
cp z ~/ECS40
```

would work.

3.10 Special Names for Some Directories

The symbol "." refers to the current directory. For example, if we are currently in Stat32 and there is a file abc in the ECS40 directory,

```
mv ~/ECS40/abc .
```

would move it to the current directory.

The symbol ".." means the directory one level up from the current one.

4 Viewing, Creating and Modifying Files

4.1 Text Editors

To create a new file or modify an old one, we use a **text editor**. A widely-used editor in Unix and other systems is **vi**. For example,

```
vi x
```

would be used to create the file x, or to modify x if x already existed.

You will need to know **vi** or some other editor in order to do many Unix operations. See my tutorial on **vi** at `http://heather.cs.ucdavis.edu/~matloff/vi.html`. You really should use one of the modern clones of **vi**, not the "plain vanilla" one. The two best clones are **vim** and **elvis**; see the above link.

4.2 Viewing the Output of a Command "Slowly," Saving It or Inputting It to Another Command

4.2.1 The "more" Command

You can view a file by using an editor, but usually it is quicker just to use the **more** command. For instance,

```
more uv
```

would display the file uv on the screen, one screenful at a time; just hit the space bar whenever you are ready to go to the next screenful. If you wish to discard the remaining screenfulls, just type q (for "quit"). If you want to back up a screenfull, type b.

4.2.2 Redirection

Sometimes you will find it useful to save the output of a command. You can do this by **redirecting** the output to a file. For example,

```
ls > y
```

will send the output of the **ls** command to a file y, instead of to the screen.[3]

Some programs have another kind of output which shows up on the screen like "ordinary" output, but which technically is considered separate. The channel through which ordinary output is sent is called **stdout**, while this special kind of output, called **diagnostic output**, is sent through **stderr**. The standard input from the keyboard is called **stdin**. You can pipe diagnostic output, say to **more** from the program **x**, as follows:

```
x |& more
```

4.2.3 Pipes

Often it is useful to **pipe** the output of one command as input to another command. Say you find the output of **ls** to be very long, zooming by on the screen before you have had a chance to read all of it. One solution to this problem would be to send the output to a file and then view the file as your leisure, but an easier, more direct method would be to pipe the output of **ls** into **more**, i.e.

```
ls | more
```

which would allow you to see the output of **ls** one screenful at a time; again, you would hit the space bar whenever you are ready to go to the next screenful.

[3] Similarly, if a command expects input from the keyboard, you can have it read from a file instead, by using '<'.

5 Online Help

You can get online information on almost any Unix command, by using **man**. For example, to get information on all the options available for the **ls** command (there is a very large number of them), type

```
man ls
```

and a detailed (though terse) description of everything **ls** does will then appear on the screen.[4] By the way, this description will be automatically piped through **more**, so as usual, just hit the space bar when you are ready to go to another screenfull.

At the end of the output, there will also be pointers to other commands related to the one requested, as well as lists of any files used by the command, such as "startup" files via which the command will have certain options set before execution.

6 The "script" Command

The **script** command is quite useful. It gives you a full record of your Unix session. For example, if you are encountering some errors which you can't fix, you could e-mail a script file to the instructor, so that you can explain to the instructor precisely what you did, and precisely what error messages you got.

To use **script**, simply type "script". Try it out as a test first: Type "script" and then after the prompt reappears, type a couple of commands, say **cd** and **ls**. Then type "exit", and a file named typescript will now exist. Look at that file; you'll see a full record of the Unix session which you just exited (your **cd** and **ls** commands, and their outputs).[5]

If you are sending a "script" file to someone for help in fixing an error, be sure to not only run the program which produced the error but also run

```
ls -l
printenv
```

so the the purpose who you are seeking advice from will know the environment in which the error occurred.

7 Leaving

To end your Unix session, simply type

```
exit
```

[4] **Note:** This is how most people learn more about Unix—by reading these "man pages," rather than say, formal lectures in a course.

[5] Don't use cursor-movement programs like **vi** within **script**, since the latter will record the cursor movements, and thus the file typescript will be almost impossible to view.