Introduction to Unix and the X Window System

Scott D. Anderson[*]
Wellesley College
Scott.Anderson@acm.org

1 Overview

Unix^{TM} is an old and venerable operating system that has also managed to grow with the times. It currently runs on almost any kind of computer, from PC hardware to big mainframes and supercomputers. Well worth learning, for any computer scientist.

The X Window System is a graphical user interface (GUI) that runs on most Unix machines. It has some nice advantages over a Microsoft Windows or Apple Macintosh GUI, primarily in being able to run over networks. Again, this is well worth learning. Indeed, a computer scientist with any breadth of knowledge would be expected to understand Unix and the X Window System. (The latter is often abbreviated to "X11" or just "X.")

There is too much about both of these topics for me to do more than scratch the surface. My goal is just to give you enough to get started and be able to run a few simple commands, write simple programs, and learn more.

2 Unix and Linux

Linux is an operating system that was intended to (1) work just like Unix, and (2) run on inexpensive IBM PC hardware (the Intel 80x86 family of chips and the PC clones). It technically isn't Unix because "Unix" is a trademarked name,[1] but you won't find any practical differences. In practice, the word "Unix" is an umbrella term that covers many variations on the Unix idea, including Linux.

Because Linux runs on PC hardware, it becomes a viable alternative to Windows NT, 95, 98, 2000, ME, XP and Vista. There are also versions of Linux that run on Macintosh hardware, so it becomes an alternative to the Macintosh operating system. Furthermore, the standard OS on the Macintosh, called OS X (and pronounced "oh ess ten") has a Unix-like core (in fact based on Open BSD, not Linux). However, the impact of Linux has been much greater in the server market than the desktop market, primarily because of its speed and reliability: Linux servers run 24/7 for months without needing to be rebooted.

Linux is also part of the Open Source movement, which means different vendors and volunteer developers share source code and build on each other's work. Thus, you can buy Linux from many different companies, or even download it for free.

The Wellesley CS department runs Linux (CentOS 6.3 as of this writing) on about 36 desktop computers in labs and offices. What I describe here runs on those computers, although almost everything runs on any version of Unix.

3 Logging In

When you come to a Linux machine, say "Wren" (most of our department machines bear animal names), there will be a login screen. You type your login name (typically the same as your Wellesley domain username) and your password. People may call this your "Puma" password, but in fact the same password works for all the machines. Your password doesn't have to be the same as your domain password and probably shouldn't be.

By default, our Linux machines run software called the "X Window System." This means that when you login, you'll get a graphical user interface that looks vaguely like Microsoft Windows or the Macintosh. There's a bar along the side called "the panel." At one end the panel is a blue icon with a stylized F on it; this is the Fedora icon. It's analogous to the "Start" button on MS Windows, and you can access many applications and configuration options from that place. There are other icons on the panel as well, analogous to the "dock" on the Macintosh. These icons are for applications that the nice folks at Red Hat (CentOS is the free equivalent of their server software) consider

[*]Thanks to LeeAnn Tzeng for helpful comments and to Agnes Ampadu for resurrecting this document that I feared was lost forever.

[1]Unix is a trademark of AT&T Bell Laboratories.

"standard." However, you can add and remove applications from the dock. There is also a little 2×2 thing called the "workspace switcher," that allows you to easily switch among four different virtual desktops.

If you right-click on the desktop, you'll get a quick menu, one item of which is "new terminal." That lets you start up a "shell."

4 Shell Commands

Back in the olden days, when you logged into a computer, there were no windows, and you typed commands to a "command line interface" and results were typed back to you (or put into files). You still get that when you login remotely using SSH. Unix still bears that legacy, and the place where you type commands is called "the shell." (It's called the shell in contrast to "the kernel," which is the core of the operating system; the shell is a command line interface to the kernel.) Nowadays, each shell runs in a window and you can have as many shell windows as you like.

The shell *prompts* you for input and responds to your commands. You can customize your prompt. Here at Wellesley, we've defined it to be your username and host (machine name) and the name of the directory you are in. If I'm logged into Wren and I'm in my `public_html` directory, my prompt would look like:

```
[anderson@wren public_html]
```

In this document, however, I will pretend that the prompt is a percent sign, just for the sake of brevity. Don't type the percent sign in the examples below. The percent sign just marks the stuff you can type.

Here are a few of the shell commands that people use every day:

ls This prints a listing of all the files in a directory (called "folders" in other operating systems). By default, it lists the files in the *current* directory, but with a command-line argument, it lists the files in the directories named on the command line..

pwd Prints the complete name of the current working directory; in other words, what the current directory is.

mkdir This creates a new directory, contained in the current one. That is, the new directory is a subdirectory.

cd This changes your current directory. In other words, it moves you from one place to another, like changing your location.

cp This copies a file from one place to another.

mv This moves a file from one place to another (or one name to another). You can rename a file by using `mv`.

rm This removes (deletes) a file. *Warning:* you can't get it back again!

rmdir This removes (deletes) a directory, but only if it's empty.

Now, let's see these in action. See if you understand what is happening at each step here. Afterward, I'll go over them and interpret. Here's a session by Wendy Wellesley:

```
% ls
HelloWorld.class  public_html
HelloWorld.java   read-only
% pwd
/students/wwellesl
% cd public_html
% pwd
/students/wwellesl/public_html
% ls
% mkdir newdir
% ls
newdir
% cd newdir
% pwd
/students/wwellesl/public_html/newdir
% cd ..
% rmdir newdir
% cd
% pwd
/students/wwellesl
% ls -1a
.
..
.bash_profile
.bashrc
.emacs
HelloWorld.class
HelloWorld.java
.kde
.login
.profile
public_html
read-only
% cp .profile dot-profile
% ls
dot-profile       public_html
HelloWorld.class  read-only
HelloWorld.java
% mv dot-profile renamed-file
% ls
HelloWorld.class  read-only
HelloWorld.java   renamed-file
```

```
public_html
% rm renamed-file
```

The first ls shows that we have two things in our current directory. The pwd shows us that the current directory is /students/wwellesl. It happens that public_html is a directory within our current directory, so we can cd into it. The pwd shows that the cd command worked. The second ls shows that the public_html directory is empty. We make a subdirectory (called newdir) and use ls to check that it exists. We can cd to it, as the pwd confirms. The Unix file structure is a tree, just like all computers, with the directories in a path separated by slashes (MS Windows uses backslashes and Mac OSX uses colons).

The cd.. changes to the directory above the current one. (The ".." is a special name for the parent directory.) This brings us back to our public_html first directory. The ls newdir shows that ls can be followed by a directory name, to list the contents of that other directory. Since newdir is empty (we just created it, after all), we can use rmdir to remove it. You can't use rmdir on non-empty directories.

The bare cd command changes to the original directory, also called the "home" directory. We confirm this with pwd. We then do an ls -al and discover that there are additional, invisible files in our home directory. The -a option to ls means to show *all* files. Usually, ls hides any files whose name begins with a dot. The -1 (digit 1) option means to give a listing in one column, which I did just for convenience in this document; in real life, -1 is rarely used. However, -l (the letter "l," for a "long" listing) is often used and gives a lot more information about each file.

We can use cp to copy one of these files, use mv to rename (or move) the copy to a new name, and finally use rm to delete the copy.

5 Successfully Copying a File

One stumbling block for many students is copying a file; you're used to graphical user interfaces, where you click and drag. The principles are the same though: specifying the source file(s) and destination folder in a hierarchical file system.

For concreteness, let's create a few directories and a file and then play around with moving that file around. Here's the setup:

```
% cd
% mkdir foo
% mkdir bar
% touch foo/a.java
```

The cd command gets you to your home directory. The next two mkdir commands create two subdirectories. Finally, the touch command creates a new, empty file named a.java in the foo subdirectory.

Now, there are actually *four* ways to specify that file.

1. /students/wwellesl/foo/a.java, which is an *absolute pathname*. An absolute pathname starts with a slash, meaning the root of the directory tree, and names each directory as it goes down the tree.
2. foo/a.java, which is a *relative pathname*. That means it starts from the current working directory (see pwd) and then names directories as it moves down. It can also move up, using .., the name for the parent directory of any directory. In this setup, we are in our home directory, /students/wwellesl, so this pathname is equivalent to the first one.
3. ~wwellesl/foo/a.java, which uses the tilde followed by a username. In Unix, the tilde looks up the home directory of someone (in a kind of database) and uses that as the starting location. Here, we'd look up the home directory of wwellesl, discover that it is /students/wwellesl and continue from there. So this is equivalent to the first two.
4. ~/foo/a.java, which uses a tilde followed by a slash. That means the home directory of the person who is logged in. Since we're logged in as wwellesl, this is the same as the previous ones.

Try the following commands, and see if they make sense. Confirm that they worked by using ls. Substitute your own username for the wwellesl.

```
% mv foo/a.java bar/
% mv ~/bar/a.java /students/wwellesl/foo/
% cd foo
% mv a.java ../bar/
% mv ../bar/a.java ~/foo
% mv a.java ~wwellesl/bar
```

Often, you're asked to copy something from a course account to your own directory, so try this:

```
% cp ~cs304/public_html/index.html ~/foo/
% rm ~/foo/index.html
```

To delete both of these directories and their contents, do the following. The -r means *recursive*, so a mistake can delete every file you own. Be careful!

```
% cd
% rm -r foo/ bar/
```

6 Creating/Editing Files

Having seen how to rename and remove files, how do we create them in the first place? This is done with an "editor," which is any program that allows you to change the contents of a file. One of the two standard editors in Unix is `Emacs`. (The other is `vi`.) I've written a separate document introducing Emacs, so I will describe it even more briefly here.

There are quite a few ways to start Emacs, depending on what effects you want. I'll just tell you just a few.

- If you're physically at the machine, called using the *console*, go to the main menu, go to the "Accessories" menu and select Emacs. That starts up a new window running Emacs. You can use the mouse to switch back and forth between any shell windows and your Emacs window.
- If you're logged in remotely via ssh, and you're on a Unix machine running X or running the X11 terminal under Mac OS X, you can start up a new window running Emacs by typing its name as a command, followed by an ampersand.

  ```
  % emacs &
  ```
- If you're logged in remotely via ssh, but you're not on a machine running X11 (say you're on a Windows machine connecting to Unix via Putty), so you only have a single shell and no desktop, you can run Emacs by typing its name as a command:

  ```
  % emacs
  ```

In Emacs, you can move around using the arrow keys, and you can type stuff into your file. You can save the file with a particular name using the "C-x C-w" command (hold down the "control" key while typing "x" then "w"). This is what other programs call "Save As." To just save a file you're working on, without specifying a name, type "C-x C-s." You can also access these commands via the "Files" menu.

I highly recommend running the Emacs tutorial. To start the tutorial, type "C-h t," which means to hold down the control key while typing "h," then release the control key and type a "t."

Note: a single Emacs can edit any number of files, so unless you have a good reason to, it's not necessary to start up multiple Emacs applications, and it will just slow down your machine unnecessarily. Instead, switch among different Emacs buffers.

7 Logging Out

When you're done with a Linux machine, don't just walk away. You must log out so that others can use the machine. If you're at the console, there is a menu item in the Fedora menu for logging out, or you can use the icon on the panel. The icon isn't easy to describe, but if you put your mouse over any icon, a balloon will pop up to give a text description, and you can find it easily enough. If you're logged in via ssh, you should use the `logout` command:

```
% logout
```

8 Learning More

To learn more about a particular Unix command, the best source is the online manual. For example, to learn more about the `ls` command, type:

```
% man ls
```

Warning: The "man" pages are reference material, not tutorials or easy reading, which means they can get a little *dense* sometimes, so skim them at first reading, and go back for more when you've swallowed that. Reading man pages is another skill that is well worth acquiring, because software changes, and even experienced computer scientists check the man pages regularly.

To learn more about Emacs, use the built-in help system. Try "C-h i" to start up the "info" pages, and then learn about that. Emacs is enormous, so don't try to learn too much at once. Start with the basics, then gradually add new things.

A web search will turn up tutorials for Unix, Linux and Emacs.

If you think of any important information or advice to add to this primer, please let me know.