L01 – Introduction to the Unix OS

1. What is Unix?

Unix is an operating system (OS): it manages the way the computer works by driving the processor, memory, disk drives, keyboards, video monitors, etc. and by performing useful tasks for the users. Unix was created in the late 1960s as a multiuser, multitasking system for use by programmers. The philosophy behind the design of Unix was to provide simple, yet powerful utilities that could be pieced together in a flexible manner to perform a wide variety of tasks.

A key difference between the Unix OS and others you are familiar with (e.g., PC) is that Unix is designed for multiple users. That is multiple users may have multiple tasks running simultaneously. Its original purpose was to facilitate software development. It is the primary OS used by physical scientists everywhere, and all supercomputing facilities use it. To put it bluntly, if you are at all on the numerical side of physical sciences, then you need to learn how to operate on a Unix OS.

In this class we are actually using a Linux OS. What is Linux? Basically the same thing as Unix. Only, Linux is developed by user contributions. Several flavors have arisen (Red Hat, Suse, Fedora, etc.) but they are all basically the same thing. What you can do in Unix you can do in Linux (the corollary of which isn't necessarily true). In this class I refer to Unix and Linux interchangeably. If I say Unix I mean Linux, and most of what I say is applicable to both.

The main difference between the two is that: (1) Unix development has corporate support. This means it tends to be a more stable OS and is the choice of those for whom stability is the top priority, (2) Linux is developed by a community of users and is free. Thus, you get what you pay for? Well, it has some stability issues and bugs creep up. But, the bugs are also quickly squashed and new content, programs, and functionality has quickly outpaced that of Unix.

I'm not going to go into detail into what the Unix/Linux OS is comprised of, but there are 3 basic entities:

1) **The Kernel** – The core of the UNIX system. Loaded at system start up (boot); manages the entire resources of the system. Examples of what it does are: interpreting and executing instructions from the shell, managing the machine's memory and allocating it to processes, scheduling the work done by the cpu's.

2) **The Shell** – Whenever you login to a Unix system you are placed in a shell program. The shell is a command interpreter; it takes each command and passes it to the operating system kernel to be acted upon. It then displays the results of this operation on your screen. Several shells are usually available on any Unix system, each with its own strengths and weaknesses. Examples are the Bourne Shell (**sh**), C Shell (**csh**), and Bourne Again Shell (**bash**).

3) **Utilities** -- UNIX provides several hundred utility programs, often referred to as commands. The commands accomplish universal functions such as printing, editing files, etc.

2. Logging into the Unix side of things

To log into the Linux side of the FASB computers **before** hitting **return** after entering your username and password select:

`Session → GNOME`

3. Getting started – really basic Unix

Now that you've logged in and opened up a terminal you are looking at a window that contains your home directory space. In case you are already confused, on Unix systems we refer to **folders** as **directories**.

Your Home Directory

- Each user has a unique **home directory**. Your home directory is that part of the file system reserved for your files.
- After login, you are put into your home directory automatically. This is where you start your work.
- You are in control of your home directory and the files which reside there. You are also in control of the file access permissions to the files in your home directory. Generally, you alone should be able to create/delete/modify files in your home directory. Others may have permission to read or execute your files as you determine.
- In most UNIX systems, you can move around or navigate to other parts of the file system outside of your home directory. This depends upon how the file permissions have been set by others and/or the System Administrator.

Unix Commands

Unix commands are programs that are supplied with the Unix OS to do specific tasks. They generally act like:

```
>> command arguments
```

Unlike your PC or Mac, instead of clicking a program icon, you type a program name in the terminal window. For example, type the following:

```
>> date
```

Date is an example of a Unix command. When used as above it simply returns the current date and time. But, we can often supply arguments to the command that modify the way the program works. For example:

```
>> date –date==yesterday
```

Here we supplied an argument asking us to return yesterday's date instead of todays.

One of the most important Unix commands is the **ls** (list) command. It lists the contents of the current directory you are in. Try the following:

```
>> ls
>> ls -l
>> ls -la
```

We can create a new directory with the **mkdir** (make directory) command. Try:

```
>> mkdir garbage
```

Now entering the **ls** command should show us that we now have a new directory called **garbage**.

We can go into this directory by using the **cd** (change directory) command:

```
>> cd garbage
```

To move back out of the garbage directory into the previous directory type:

```
>> cd ../
```

Note that we can go back multiple directories if we want to:

```
>> cd ../../../     (etc.)
```

Where the **..** always stands for the previous directory. After moving around directories it can get confusing as to where you are. So use **pwd** (print working directory):

```
>> pwd
```

to see where you are. You can always go right back to your home directory by typing either:

```
>> cd ~username
```

or just

```
>> cd ~
```

or even just

>> **cd**

The primary reason to use the tilde (~) is so that we can back to directories starting from our home directory. (e.g., **`>> cd ~/Utilities/`** if the **Utilities** directory was located in my home directory)

Perhaps your sick of your directory called garbage. You can get rid of it with **rmdir** (remove directory):

```
>> rmdir garbage
```

We can also make files. We will talk more about this now, but let's just try the following:

```
>> echo "I love geophysics" > geophys.txt
```

```
>>  echo "I really love geophysics" >> geophys.txt
```

The **echo** command just echo's whatever you write, and in this case redirected "I love geophysics" into a text file called **geophys.txt**. Perhaps I wasn't happy with the file name I created and wanted it be named **Geophys.txt** (note that we are working in the C Shell and that file names are case sensitive), then I could use the **mv** (move) command:

```
>> mv geophys.txt Geophys.txt
```

Or maybe I wanted another copy of this file **Geo.txt** called **Geo_copy.txt**

```
>> cp Geophys.txt Geo_copy.txt
```

You can guess already that the **cp** command means copy. As you can tell, there are a lot of Unix commands. The examples shown above are some of the most important, but are really just the tip of the iceberg. The following web page shows what some of the most important basic commands are:

http://mally.stanford.edu/~sr/computing/basic-unix.html

Special Characters

There are also some very special characters that you can type in Unix. The next table shows just a few of them:

Character	Function
*	wildcard for any number of characters in a filename
?	wildcard for a single character in a filename
$	References a variable
&	executes a command in the background
;	separates commands on the same line
>	redirects standard output

From the above example we should still have two files around named **Geo.txt** and **Geo_copy.txt**. What if I want to see all of the files I have that have a name starting with **Geo**? I can use the * character:

```
>> ls Geo*
```

Or just files with the word **copy** in them?

```
>> ls *copy*
```

We will introduce more of these characters later.

Getting Information on Commands

Most Unix commands have several options and can be used in a variety of ways. To get full instructions on a Unix command there is the **man** (manual) utility. For example to see all of the ways you can use the **ls** command type:

```
>> man ls
```

Logging Off the System

To log off the system select the **Red Hat** icon in the lower left hand corner of the screen. Choose the **Log Out** option.

4. Editing Files

One of the most important choices you will make in learning Unix is what text editor should you use. This is likely not a question many of you anticipated, as on a Windows or Mac one rarely ever uses a text editor – unless you refer to Microsoft Word (it is a text editor, but how many times do you store data in **.txt** format?). On a Unix system there are several choices of editors. Peoples defense of their choice of editor is similar to a religious conviction, so be careful in talking bad about other editors. Two of the most popular choices of editors are:

- **vi** – (pronounced *vee – eye*) one of the earliest advanced editors, was installed on every system. *vi or die* was a common expression, as it was the only editor found on many systems.

- **emacs** – a more recent editor, many people prefer this one, and it can now be found as commonly as vi. Emacs has many more commands than vi.

You can use whatever text editor you choose. The choice is yours, but you are responsible for learning one on your own. Learning to use an editor is not a choice though. This is mandatory if you want to be successful in computation. I have heard not knowing an advanced editor described as "being like a car without an engine under its hood." I personally like **vi**, which is reportedly rather difficult to learn at first. But, I can maneuver around a file in **vi** way faster than any other editor so here's a super fast intro:

To create a new file, or open an old file type:

```
>> vi myfile.txt
```

The key thing to remember is that **vi** has two modes: **command**, and **insert**.

When you are in **command** mode, everything you type on the keyboard gets interpreted by **vi** as a command. Command mode is the mode you start out in.

Now that you are in your file enter into insert mode by hitting the **i** (for **i**nsert) key. You should notice that at the bottom of the screen it now says you are in **-- INSERT --** mode. Now anything you type shows up on the screen.

When you are in insert mode, you can switch back to command mode by pressing the **Esc** key on your keyboard.

When you are in command mode, there are many keys you can use to get into edit mode, each one gives you a slightly different way of starting to type your text. In addition to insert there is also **a** for **a**ppend, **o** for **o**pen a line, etc.

To wrap up a **vi** session, hit the **Esc** key to get back into the command mode. Now you save the file by hitting **Shift+ZZ**.

A good tutorial can be found here: http://www.rru.com/~meo/useful/vi/vi.intro.html

A **vi** reference card is located here: http://limestone.truman.edu/~dbindner/mirror/vi-ref.pdf

5. A few more important commands

Now that we know how to create files what are the basic ways we can access their contents. To start out everyone create a file called **temp.txt**:

```
>> vi temp.txt
```

hit the **i** key for insert and type some words, for example here are some nice words from Edward Abbey's famous book Desert Solitaire:

"The love of wilderness is more than a hunger for what is always beyond reach; it is also an expression of loyalty to the earth, the earth which bore us and sustains us, the only home we shall ever know, the only paradise we ever need – if only we had the eyes to see. Original sin, the true original sin, is the blind destruction for the sake of greed of this natural paradise which lies all around us – if only we were worthy of it."

Now hit the **Esc** key and save the file by hitting **Shift+ZZ**.

If we do an **ls** we can see that our file **temp.txt** now exists. But this doesn't tell us anything about what is in the file.

Viewing Files:

There are several ways we can see the contents of the file. Try the following commands:

```
>> cat temp.txt
```

```
>> less temp.txt
```

So, what was the difference between the two commands?

Now try the following:

```
>> head -1 temp.txt
```

```
>> tail -1 temp.txt
```

These commands are obviously useful if you want to see the top or bottom of a file. What if we want to know something about the file like how many words does it contain? Look up the man page on **wc** (word count) and find out how you can (a) determine how many words the file contains, (b) how many characters the file contains, and (c) how many lines the file contains.

(a)

(b)

(c)

Another really useful command is **grep**. This allows us to search files to find specific instances of words. For example, we could say let's just find the lines in temp.txt that contain the word paradise.

```
>> grep paradise temp.txt
```

Grep is really useful when I'm searching for something specific in a lot of files.

Zipping and Unzipping Files:

It's important to understand this as most of the files you will download off the webpage for this course are zipped. There are, as is typical, many choices of zipping utilities. Generally we use the utility called **gzip**. To zip up, or compress, our file **temp.txt** simply type:

```
>> gzip temp.txt
```

Now if you do an **ls** you will see the filename is changed to **temp.txt.gz**. Note that the **.gz** extension will often be found on files you get from me. This means they have been compressed with **gzip**. What happens if you try and view the contents of this file with **cat temp.txt.gz**?

Right, to view its contents we need to unzip or uncompress it:

```
>> gunzip temp.txt.gz
```

You will also notice that most of the files you download from me have the **.tar** extension. These tar files stand for **tape archiving** (still in use for backups today!). Usually we use tar to lump a group of files together into one single file. Then we only need to send one file and not a bunch. To see how tar works let's do as follows:

```
>> cp temp.txt temp_copy.txt

>> mkdir TempFiles

>> mv temp*.txt TempFiles

>> tar cvf TempFiles.tar TempFiles
```

Now you will notice there is a file called **TempFiles.tar**. The **tar** command we used in the above example used the flags to **c**reate a **f**ile called **TempFiles.tar** from the directory and its contents **TempFiles**.

Most of the files you download from the webpage will have the **.tar** extension. To unpack these files:

```
>> tar xvf TempFiles.tar
```

Where now we used the e**x**tract flag.

Job status:

It is also useful to see what is currently running on the computer you are using. The quickest way to do this is to use the **top** utility. Just type:

```
>> top
```

But, you can get specific information using the **ps** (processes) utility. E.g., to see what programs you personally are using type:

```
>> ps –u username
```

Where you fill in **username** with your personal username. This is especially useful if you've started a bunch of jobs or maybe someone else did on your computer and its eating up the cpu or memory. Notice that all jobs have a number associated with them under the column **PID** (Process ID number). This number is important. **Don't** actually do this now – but in the event that you absolutely need to stop something that is running you can do this with the **kill** command:

```
>> kill -9 PID
```

This will force the process, whatever it is to be stopped. So only use this if you absolutely need to stop the job and you know what the job is.

6. Customizing your environment – the .cshrc file

Before we wrap up this intro lets talk about your **C Shell Resource File** or **.cshrc** (some people also call it a C Shark file but that drives me crazy so please don't use it). This file is really important because it gets read by the Unix system every time you log in or every time you open up a new terminal window.

This file lives in your home directory, so change directories to your home directory and let's look at its contents:

```
>> cd ~
```

```
>> less .cshrc
```

There are two main things I want to point out in this file: (1) your search path, and (2) aliases.

Search Path

When you type a Unix command at the command prompt (e.g., **cd** or **ls**) the Unix Shell looks for a program with that name. Things like **ls** or **cd** or **mkdir** are all programs that reside in a directory somewhere. For example, if you want to find out where the **ls** command lives type:

```
>> which ls
```

So, on the computer I am working on as I write this document, I see that **ls** is located at: **/usr/bin/ls**. Or, it lives in the directory **/usr/bin/**. For the Unix system to be able to execute the **ls** command it has to be in the **Unix Search Path**. That is, Unix has a special variable called **PATH** that contains a collection of directory names to search through for commands that are typed. To see what directories Unix is currently searching through for you type:

```
>> echo $PATH
```

OK, but what if I make a program that I want Unix to be able to use (and believe me you will!)? The most common thing to do is to create a special directory where you will store your personal programs, then add that directory name to the **PATH** variable.

I put all of my personal programs into a directory called **Utilities/bin**. So, you could do the same:

```
>> mkdir Utilities
>> cd Utilities
>> mkdir bin
```

Now we need to add this directory to the search **PATH**. We do this by adding a line to our **.cshrc** file:

```
>> cd ~
>> vi .cshrc
```

now go down somewhere to the bottom of the page and **i**nsert:

```
set PATH = ($PATH ~/Utilities/bin)
```

Save the file with **Esc** then **Shift+ZZ** sequence.

After you've saved the file, we need to tell the Unix system to re-read our **.cshrc** file. We do this by typing:

```
>> source .cshrc
```

Aliases

Another fine use of the **.cshrc** file is to create aliases, or shortcuts. For example, instead of just the normal **ls** command I like using the following flags: **ls -F -h --color=always**. So, I can add a line to my **.cshrc** file that says every time I type **ls**, actually do: **ls –F –h –color=always**. We can do this by adding the following line to our **.cshrc** file:

```
alias ls="ls -F -h --color=always"
```

Another favorite of mine is to just be able to type net to launch an internet browser:

```
alias net="mozilla &"
```

I also like the fancy printing style that comes out of:

```
alias lpt="a2ps -o- -d --medium=letter"
```

7. Homework

Probably the most important thing you can do this week is start getting a good handle on a text editor. Hence, I want to you pick a text editor and practice using it. If you do not do this you will quickly fall behind in this class in a manner you will not be able to recover from. So, your homework is:

Choose a text editor and create some files. Create one file and tell me: (1) what your major is, (2) if you are a graduate student tell me who you are working with and what your research project is about or if you are an undergraduate tell me what your plans are after graduation, and (3) what you want to get out of this class. Also, if there is a special computational task or tool you want to learn in this class that isn't currently on the syllabus please tell me what it is and why its important for you to learn that.

Create a second file tell me which editor you chose to use and what your experience is about learning it.

Now make a directory and move these files into that directory. **Gzip** the files and make a **tar** file of the directory with the files. Create the tar file with the following naming convention:

Lastname_Firstname_HW1.tar.

Copy that file to a location on my personal home space. That is, copy the file to:

```
>> cp Lastname_Firstname_HW1.tar ~mthorne/GG5920_HW
```

All of your homework will be turned into me this way.