Unix Quick Start – Basic Unix commands to get you started.

If you have only ever used a Windows operating system, or the point-and-click features of your Mac OS-X operating system, then learning to use Unix will take a dedicated, but reasonably small, effort. While the Unix Common Desktop Environment (CDE) has some point-and-click features, you may not be running the CDE if you are connecting remotely into a Unix server. Instead, you will be operating in your Unix environment by typing commands at the command prompt in a terminal window. This Quick Start Guide will provide you with a dozen or so Unix commands to get you operational in Unix. You do not need to understand every facet of Unix to be operational in Unix. A majority of tasks can be completed by only knowing one to two dozen Unix commands and a little background knowledge on the correct way to perform some very common tasks.

(0) Is it Unix or Linux?

In this document we are using the term Unix generically to refer to both Unix and Linux. For our purposes we can think of Unix and Linux as commercial and open source cousins. Unix is the original commercial operating system that is currently trademarked by The Open Group, and Linux is the open source cousin, which has probably a dozen different 'flavors', called distributions. Common flavors of Linux are Red Hat Linux and its open source counterpart Fedora, open source Debian, and the commercial IBM AIX version. By simply using the term Unix in this document, we are trying to keep it simple. While Unix and Linux commands differ, some differ slightly and some differ more than slightly, the topics that we want to discuss in this document will be identical or almost identical. If you want to further pursue the areas of administration or scripting, then you will want to pay attention to the details between Unix and Linux, e.g. buy a 'Linux' book, not a 'Unix' book, and you will also want to pay attention to the details of your particular Linux distribution.

When reading and using this document, it is important to remember that this document is not all inclusive. This document is designed to give you a Quick Start to using Unix, hence the document name. To learn more about Unix, you can search the web or buy a book like *Unix in a Nutshell* or *Linux in a Nutshell*.

(1) What does it mean to remote into a Unix server?

When we remote into a Unix server, we will be connecting to a Unix server from our personal computer using a SSH (Secure Shell) client. This connection will require a SSH software, and typically a X-windowing software, to be installed on your local machine. The connection process is very similar to using a file transfer client like Filezilla, which is a SFTP software (Secure File Transfer Protocol). Once we have connected to our remote Unix server by SSH, we will typically run a X-windowing software in order to allow us to *tunnel* applications to our desktop. Tunneled applications are much easier to use, as they do not require us to operate them directly in the terminal window. As an example, we have the option of running a software such as SAS by submitting a SAS program directly from the command line in our Unix terminal or by tunneling the application and allowing our X-windowing system to display the SAS GUI.

- Typical X-windowing systems are X-Win32 for Windows and X11 for Mac OS-X.

(2) What does a terminal window look like?

If you have a Mac, then you have the option of working directly in a terminal anytime that you wish. The Max OS-X operating system is a Unix based graphical interfacing operating system. Simply click on the terminal icon in your toolbar, and a terminal window will appear.

If you are accustomed to the Windows operating system, then you will associate the terminal window with your Command Prompt, or your old DOS terminal. Fortunately, a Unix terminal is a lot nicer and easier to use than its Windows cousin.

(3) In order to write programs on a Unix server you need a text editor.

You have the choice of Emacs or Vi as your text editor. Both applications have versions that will tunnel and create a stand alone window for you to type in. You will need to decide which one of these text editors you will use.

- Tunneling versions of Vi will have names like Vim or eVim.

(4) When working in Unix you need to organize your files.

In Unix you will need to organize your files by creating *directories* or *sub-directories* inside your *home directory*. On your graphical interfacing operating systems you do this by creating *folders*. Here, you will create 'directories' and you will use the Unix command **mkdir** to 'make directory'. This directory structure has a fancy name called a *hierarchical file system*. The name does not affect us much, but we want to make sure that we are only making directories within, or better yet – underneath our home directly. The typical user should not be making directories in other areas of the file system, and the file system should have a set of user permissions that will not allow you to operate in areas outside of your home directory.

(5) Finding help on Unix – the *man* pages.

The Unix help files are called the *man* pages, which is an abbreviation for 'manual' pages. The man pages can be searched by command or by keyword. We search by command when we need to understand the syntax required by a particular command or when we want to know what options are available as *flags* for a command.

<u>Example1:</u> Search by command – need to understand how to use **mkdir.**

```
:> man mkdir
```

<u>Example2:</u> Search by keyword – need to find a program to open a pdf.

```
:> man –k pdf
```

<u>Example3:</u> Man the man page – display all of the flags available for the **man** commend.

```
:> man man
```

(6) What Unix shell are you using?

I recommend that everyone use the Bash (Bourne Again) shell. If you are using a form of Linux, like on your Mac OS-X operating system, the default Linux shell is bash. Most academic installations use Bash as the default shell, but corporate installations may not, and some will not. For example the default shell for the IBM AIX Linux operating system is the Korn shell (ksh). No matter what shell is set as your default shell, you can always invoke the Bash shell by typing **bash** at the command prompt.

Find your default shell by using one of these commands.

```
:> echo $SHELL

:> ps –p $$

:> echo $0
```

If the Bash shell is not your default shell, then you can invoke the Bash shell from the command line by typing **bash**.

```
:> bash
```

The Bash shell has simple, but useful, features that are not available in all shells, specifically Korn shell. These features include auto-complete and command toggle. The command toggle will let you scroll through your previous command line entries by using the up and down arrow keys.

Unix Commands and Their Uses

Managing Directories:

(1) Make a directory to store your files.

```
:> mkdir Predict_410
```

Note that we do not leave spaces in file or directory names in Unix.

(2) Move down the directory tree into your new directory – change directory, and create a subdirectory for your assignment.

```
:> cd Predict_410
:> mkdir Assignment_01 Assignment_02
```

Here we have made two directories using one command.

(3) Create additional subdirectories to store the code and data for Assignment #1.

```
:> cd Assignment_01
:> mkdir Code Data
```

(4) Move back up the directory tree one level.

```
:> cd ..
```

The double period means the directory up one level on the directory tree. This will move you from the Assignment_01 directory to the Predict_410 directory that contains Assignment_01.

The single period means the current directory. We will use this notation later in this tutorial.

(5) Move to an entirely different branch in your directory tree.

```
:> cd /home/userid/Predict_410/Assignment_02/
```

Note that we are assuming the name of the path. Your actual path will differ from the example, but it will generally have this form. By listing the whole path you can move directory from the Assignment_01 directory to the Assignment_02 directory without moving up and down the tree. You simply move across the branches of the tree by specifying the full path.

Managing Files:

(1) Print your working directory.
:> **pwd**

This command is an acronym - **p**rint **w**orking **d**irectory, and hence it is very easy to remember.

(2) List the files in your current directory.
:> **ls**
:> **ls** ./
:> **ls –l**

The –l (dash L, not dash one) is a flag for 'list long' format. See the man page for other flags (options) available for the **ls** command.

Note – Unix is case sensitive, hence it is important that we use –l and not –L. It is also important that we recognize that myfile.txt is a different file from myFile.txt.

(3) List all SAS files in your current directory.
:> **ls** *.sas

Here * is the wildcard value.

(4) List the hidden files in your directory.
:> **ls –a**

Many Unix utility files, or application files, start with a period. Some common examples are your .profile, .bashrc, and .emacs files. Files that start with a period are called hidden files, and they will not display in the **ls** command without the **–a** flag ('list all'). Most hidden files are parameter files, files that control the settings on your user account or your applications, and hence they usually only exist in your home directory. Use the **ls –a** command in your home directory to see the hidden files attached to your user profile. Do not modify these files unless you know what you are doing. If you were to modify one of these files, always first create a backup file.

:> **cp** .bashrc .bashrc_backup

(5) Create a copy of a file.

```
:> cp file1.txt file2.txt
```

Here we have created a copy of file1.txt and named it file2.txt.

(6) Move or rename files.

```
:> mv file2.txt file22.txt
```

Here we have renamed, or 'moved', file2.txt to file22.txt. The file file2.txt no longer exists.

(7) Delete or remove a file.

```
:> rm file1.txt
:> rm file*
:> rm *.txt
```

Here we have deleted or 'removed' a single file by its full name, multiple files that start with the letters 'file', and multiple files that end with the file extension .txt.

(8) Create a copy of a directory.

```
:> cp –R Assignment_01 Assignment_01_backup
```

Here we have created a copy of the directory Assignment_01 and all of its contents. The **–R** flag is the 'recursive' flag, and it will apply the copy command recursively to each item in the contents of the directory Assignment_01.

(9) Delete or remove a directory.

```
:> rmdir Assignment_01_backup
:> rm –R Assignment_01_backup
```

If the directory Assignment_01_backup is empty, then it can be removed by using the 'remove directory' command **rmdir**. If the directory is not empty, then it must be 'removed recursively' by the command **rm –R**.

User Groups and File Permissions:

(1) Unix user groups.

When you receive your Unix account, the Unix administrator may or may not include you in a set of user groups. User groups are an integral part of the Unix file system. By assigning users to user groups, users are allowed to distinguish, or partition, file permissions amongst the other users on the basis of their group membership – either they are in your user group or they are not.

(2) Managing file permissions.

Argument codes needed to manage your file permissions using the **chmod** command.

u	user – you
g	group – assigned to you by the Unix administrator
o	others – everybody else on the server that is not in your group
+	add permission
-	subtract permission
r	read
w	write
e	execute

<u>Example1:</u> Add read and write access to a specific file for 'other users'.

```
:> chmod o+rw group_assignment.sas
```

<u>Example2:</u> Add read and write access to a specific directory and all of the files in that directory for 'other users'.

```
:> chmod -R o+rw Assignment_01
```

<u>Example3:</u> Lock a directory down so that no one can read or write to it.

```
:> chmod -R go-rw Assignment_01
```

Managing Processes:

(1) List out all of your active processes.

```
:> ps
:> ps –u userid
```

The **ps** command will list out your processes. Likewise the **ps –u** command will list out your processes if given your userid. However, you can also list out another user's processes if you provide their userid.

```
:> ps

     PID    TTY  TIME CMD
  9240642 pts/18  0:00 -ksh
  9502824 pts/18  0:01 dtwm
 11141224 pts/18  0:00 /usr/dt/bin/dtterm
 15859892 pts/18  0:02 /usr/dt/bin/dtterm
 19791874 pts/18  0:00 /usr/dt/bin/dtterm
 27984102 pts/18  0:00 /usr/dt/bin/ttsession -s
 29360236 pts/18  0:00 /usr/dt/bin/dtsession
 30736602 pts/18  0:04 /usr/dt/bin/dtterm
 31785098 pts/18  0:00 ps
 36110510 pts/18  0:00 bash
```

In this table of output we are primarily interested in the Process ID (PID).

PID:	Process ID number
TTY:	controlling terminal of the process
TIME:	CPU time used by the process
CMD:	name of the command that initiated the process

Note that the output from the ps command will differ from one Unix distribution to another. These fields will always be included, although their names might be slightly different.

(2) Terminating a process.

If a process is running in the terminal, such as from the **man** command or the **more** command, then that process can be terminated or killed by using the key sequence Contol-x, Control-c. If the process is running in the background or through a tunneled application, then the process must be killed using the **kill** command with the process id. Note that multiple processes can be killed at one time.

```
:> kill PID
:> kill 9240642 9502824
```

If a process does not respond to the kill command, then include the **-9** flag to the command.

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

Some Input/Output and Utility Functions:

(1) Direction, Redirection, or Putting
Easily used to put the output of a Unix command into a text file.

```
:> ls –l > myFileList.txt
```

Here we put the list of files in our current directory into a text file.

(2) View a text file in the terminal – the **more** command.
We can view, or preview, any text file in the terminal by using the **more** command.

```
:> more myFileList.txt
```

Here we can view our file list in the terminal with the paging convenience provided by the **more** command.

(3) Piping – Deliver the output of one function to another function through a pipe.

If we did not want to save our file list to a text file, but we still wanted to view it using **more**, then we could pipe the results to **more**. The vertical bar | is called ‘the pipe’.

```
:> ls –l | more
```

(4) Searching with **grep**.

The **grep** acronym stands for **G**lobally search a **R**egular **E**xpression and **P**rint. The **grep** command is a workhorse command as you get more and more serious about Unix and shell scripting. In this tutorial we are only going to introduce you to the grep command and some common uses.

Example1: List out files that contain the string '20140201' by piping the list command.

```
:> ls | grep '20140201'
```

Example2: List out the files that contain the string '20140201' or '2040301' by piping the list command. Here we will use the sister function **egrep** – **E**xtended **G**lobal **R**egular **E**xpression **P**rint.

```
:> ls | egrep '20140201|20140301'
```

Example3: Search a text file for lines containing '20140201'.

```
:> ls –l > myFileList.txt
```

```
:> grep '20140201' myFileList.txt
```

Example4: Search a text file for lines containing '20140201' or '20140301'.

```
:> egrep '20140201|20140301' myFileList.txt
```

(5) Comparing two files – the **diff** command.

```
:> diff new_file.txt old_file.txt
```

The **diff** command will directly compare two files and highlight their differences.