Bash/Unix Command Cheat Sheet

Leaning how to utilize the command line in Unix is crucial for navigating your computer as well as making, deleting, and searching your files. Many of these commands will come in handy on a very regular basis, so they should become second nature rather quickly. Until then, feel free to use this sheet!

Before you get started:

```
% cp ~premapta/setup ./
% tcsh setup
```

The above command will set up some directories as well as your .cshrc file. This file is important, as it has in it settings that will point your computer where to important files. We will do this together in class and it only needs to be done once.

Some General Notes and Definitions

I will be using the term “**directory**”. Think of these as files, or places on your computer. For example, your desktop is a directory. A folder on your desktop is a directory *within* the desktop directory. etc, etc....

Your “**home**” directory is your starting place. Whenever you first open up the terminal, you start from here. Often, this is denoted by a “~”. To go back to your home directory:

`% cd ~` or even just `% cd`

An object’s “**path**” means its location on the computer (think of it as an address that tells you exactly where something is). For example, a file named “file.txt” in the directory “text_files” in the folder “MyStuff” on your Desktop would be:

```
~/Desktop/MyStuff/text_files/file.txt
```

Think of your **path** like a stream. Your home directory is the highest point and everything flows out from there (i.e. everything is “down stream”). Everything within a directory is “down stream” from that directory.

Any command that takes a file or directory name as input can also take a path as input. For example, the following lines do the same thing (they remove the file called “file.txt”)

```
% rm file.txt
% rm ~/Desktop/MyStuff/text_files/file.txt
```

When you want to perform an action on multiple files in a directory, using “ `*` ” is useful.

`% rm *`	removes all files in a directory (dangerous!)
`% rm *.txt`	removes all files with names that end in ‘.txt’

`% rm test.*`	removes all files with names that start with “test.”
`% rm test*.txt`	removes files with names that start with “test” and end with “.txt”

Many of the commands discussed below have options connected with them. Options are usually denoted by upper or lowercase letters (and sometimes numbers) after the command, usually preceded by a ‘-’. For example,

```
<command> -[option1][option2]
```

If you forget the exact syntax to use with a command, or need to find what options are available, use the `man` command. For example:

```
% man ls
```

This will give you the manual page for the `ls` command, including a list of the available options and their actions.

Tab completion is very useful. When you are typing in a command or the name of a file, hitting tab will have the computer attempt to auto complete what you were typing based on the available options. If there is more than one option, it will give you a list of the possible commands/files. For example, type “`ip[TAB]`”. The computer gives you a list of all files, directories, and commands that start with “ip”. Type in “`ipy[TAB]`” and the computer auto completes the command to “ipython”.

Listing files in a directory

`% ls`	Gives a list of (almost) everything in your current directory
`% ls -a`	Lists everything in the directory (including files that start with a ‘.’)
`% ls -l`	Lists Things in an up-down list with some information, like when the file was last edited and the size of the file
`% ls -la`	Does both options!

Navigating directories

`% cd [directory path]`	Moves you to a new directory

`% cd ../`	Moves you “up” one directory
`% cd ../../`	Moves you “up” two directories
`%pwd`	Shows you where you are (i.e. your current directory’s path)
`% which [command]`	Tells you the location of the executable you are actually running when you type in a command.

Making and moving files and directories

`% mkdir [directory name]`	Make a new directory within your current directory
`% rmdir  [directory name]`	Remove a directory with the specified name. *This will only work on empty directories*
`% rm  [file name]`	Remove a file with the specified name. *Will not work on directories*
`% rm -rf  [file name]`	Removes a file with the specified name. This WILL work on directories that are not empty.
`% cp [file1] [file2]`	copy file1 to file2
`% mv [file1] [file2]`	moves file1 to file2. This can be used to move the location of a file or it can be used to rename files.
`% wget [webaddress of file]`	Download a file from the internet into your current directory

Writing and Reading Files

`% echo hello`	prints out “hello” to the screen

`% echo hello > [file]`	opens a file and prints “hello” to it. This will overwrite the file and delete whatever was in it at first
`% echo hello >> [file]`	Same as above, but it will append the file rather than overwrite it
`% cat [file]`	Print out the contents of a file
`% cat [file1] > [file2]`	Print the contents of file1 to file2 (overwriting file2)
`% cat [file1] >> [file2]`	Append file2 with the contents of file1
`% grep [something] [file]`	print out the lines in file that include the string “something” in the line
`% more [file]`	View the contents of a file. Better than `cat` if the file is long. Push spacebar to scroll down
`% tail -[number] [file]`	View the last [number] of lines of a file
`% head -[number] [file]`	View the first [number] of lines of a file
`% wc [file]`	Prints number of lines, words, and size of file

Sorting Streams

`% sort -k number [file] | more`	Alphabetical sort, where number refers to the column.
`% sort -n -k number [file] | more`	String numerical sort.
`% sort -g -k number [file] | more`	General numerical sort.

Making “tar balls” (compressed files)

`% tar cfz filename.tar.gz [list of files]`	Compress files into filename.tar.gz. (note the lack of a “-” before the cfz option here....)
`% tar xfz filename.tar.gz`	Decompress filename.tar.gz and put all the files in it into your current directory

Some examples using “**piping**” (“|”)

`% cat [file] | grep word`	Print the lines in file that include “word” in the line
`% grep word [file] | wc`	Gives the number of lines, words, and size of the portion of file that grep returns as having “word” in the line

Checking the memory usage on your machine

Sometimes you will run commands that will push your machine to its limit. Or, you might be using a shared device and want to avoid hogging all the memory for yourself. To check what processes are being done by your machine type in the command

```
%top
```

To only look at processes you are running, type

```
%top -u [username]
```

This will cause some information to pop up on your terminal. For example:

```
top - 09:05:39 up 23 days, 21:18,  7 users,  load average: 0.10, 0.04, 0.00
Tasks: 223 total,   1 running, 222 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.4%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16247332k total,  2899880k used, 13347452k free,   162968k buffers
Swap:  8388600k total,        4k used,  8388596k free,  1420460k cached
```

The first couple lines are pretty useless.

The line starting with “Cpu(s)” tells you how your computer is using its CPUs. Really, the only thing relevant here is that if you are running something, if it isn’t using 100% of the CPUs, then it is technically running inefficiently (though I would say this usually doesn’t warrant any worry).

The next line is very important. This tells you how much RAM (memory) you are using (in kilobytes). You want to avoid doing things that push your computer to its limit, so be mindful of this. “Used” should always be less than “total”.

Finally, the “swap” line can be important. Don’t worry too much about what this means, just remember that you DO NOT want to be using swap memory. If the “used” swap memory is INCREASING, then you have a problem on your hands. This is a result of you using too much memory.

Typing “q” will quit out of the top display.

Logging onto machines from home

Step one: open up a terminal (PC users: download Google Chrome and get the Secure Shell add on). Remember from class that we need to do some basic setup. Type the following command to get the correct “config” file on your laptop:

```
% scp UWNETID@astrolabXX.washington.edu:~/.ssh/config ~/.ssh/
```

where “UWNETID” is your username and [XX] is numbers ranging from 1 to 29
Type in your password when prompted. Open up the config file in a text editor and put in your UWNETID where it says UWNETID and type in the astrolabe computer you want to connect to where it says astrolabXX. Now you should have the proper config file, and in order to remotely log in all you need to type is the following:

```
% ssh astrolabXX
```

Again, enter your password when prompted (will probably be prompted twice). Now you should be at your home directory on the astro computer! If you want to log out at any point just type “exit” in the terminal.

Use this when working from home so that all your work stays on the computers here (also, some programs cannot be used outside of this lab)

Copying things between machines

To do this, use the **secure copy** command, scp.

In general:

```
% scp [files you want to copy] username@<other_place>:<path>
```

You will be prompted for your password before the copying will take place

A specific example:

```
% scp file premapta@astrolab23@astro.washington.edu:~/python/pro
```

The above line copies the file “file” to the directory python/pro within the account of “premapta” on the astrolab23 machine. Recall from above that in order to copy something FROM an astrolab machine TO your laptop you would do something like the following:

```
% scp premapta@astrolab23@astro.washington.edu:~/python/pro/
file .
```

Where the period at the end just means to copy that file to whatever directory you are currently in (you could put in a more specific path).