Make sure you use < tab> to autocomplete words! Using <tab><tab> gives you a listing of all files/directories that match the autocomplete.

**List contents of a directory:**

ls –l (long description)

ls –t (sort of time)

ls –S (sort by size)

ls –r (reverse sort – in combination with –t or –s)

(Remember you can always “man” a command to get more options)

**change directories**

cd ../the/path/to/your/directory/

cd .. (moves you back a directory)

**move a file**

mv somefile.txt ../newdirectorypath/

**remove (delete) a file**

rm some file.txt

**remove a directory**

rm –r ../directorypath/ #note this a dangerous command, since it will delete all files and subdirectories

the “-i” option will make you confirm the deletion of files and is a good one to use as a beginner.

**Wildcard characters:**

? (represents a single character) 🡪 rm test?.txt (this would remove test1.txt, test2.txt, testX.txt)

\* (represents all characters following) 🡪 rm test\* would remove any file with starting with test, regardless of the following characters. BECAREFUL with wildcards.

**rename (uses the mv command)**

mv somefile.txt newnameoffile.txt

**copy**

cp somefile.txt somefile\_2.txt

with wildcards: cp somefile\* .. (copy all files that start with somefile up a directory).

**Concatentating files**

cat file1.txt file2.txt > file3.txt

Example:

Cat \*.txt > all.txt

**Identifying your current path**

pwd

**History command lists all the commands you’ve used**

history (prints a list of all previous commands)

#you can also get to your history one command at a time by simply using the up arrow on your keyboard. Meaning – if you want to rerun the command you just typed, don’t retype it, use the up arrow to find it and press <enter>.

**Keyboard navigation of the command line:**

Ctrl + a 🡪 move to the start of a line

Ctrl + e 🡪 move to the end of a line

**Clear screen (for neatness!)**

clear

**Creating a file (this will create a file – although it will be empty)**

touch somefilename.txt

Or

vi somefilename.txt 🡪 this will open up an editor called vi. You can invoke editing mode by pressing <i>. This will also you to insert text. Then you must escape from the insert mode by pressing <esc>. You must them “write” the file by typing “:w” and then you can “quit” the vi editor program by typing “:q”. BTW you could just type “:wq”.

**Viewing a file**

less somefilename.txt (this will print the file to the screen, use <q> to get out of the program)

h 🡪access to help

g 🡪jump to start of the file

G 🡪jump to end of the file

/someword 🡪 search for “someword” (search is only forward)

?somework 🡪 search for “somework” (searc backward in the file).

q 🡪 quit

#NOTE: less is not a text editor!! It’s just for viewing files quickly.

**Viewing the beginning and end of a file:**

head somefile.txt

tail somefile.txt

use the –n to change from the default number of 10 lines to show

**Edit a file using vi**

vi somefilename.txt 🡪 this will open up an editor called vi.

You can invoke editing mode by pressing <i>. This will also you to insert text.

Then you must escape from the insert mode by pressing <esc>. You must them “write” the file by typing “:w” and then you can “quit” the vi editor program by typing “:q”. BTW you could just type “:wq”.

**Search within vi:**

/someword

**Global search and replace within vi:**

:%s/someword/somenewword/g

**Changing unix permissions (remember perl programs must be executable to use!)**

chmod 777 – gives complete read, write and execute to owner, group and world

chmod 775 - gives complete read, write and execute to owner, group but only read and execute to world – this is probably the one that you’ll use the most.

**Modifying your $PATH**

Anytime you try running a program, unix will check through a list of predefined directories to see if the program exists. Often times you will want to add your programs to the this list of directories so you can run your programs from anywhere in your directories – if not you’ll need to specify the specific path to the program each time you run it. However if you put all your code in a directory called “code” we can direct unix to all look here for any programs we want to launch from the command line:

Check your PATH by:

echo $PATH

**Temporarily add directories to your $PATH:**

export PATH=$PATH:/path/to/your/codedirectory

#remember that if you don’t remember the exact path to your code directory, you can also navigate there and use the pwd command to get that path

Check to see if was added to your PATH using the echo command (echo $PATH)

Each time you open up a shell, you’ll need to type this export command – its temporary!

**Permanently adding directories to your $PATH:**

To make this permanent, so that everytime you open a shell, it knows that command are in your code directory, you need to change your .bash\_profile file. The .bash\_profile file contain the start up information for the shell (terminal). To add to the .bash\_profile file you must first find it (and it’s hidden!).

First go to your home directory.

cd (cd by it self will bring you to your home directory – mine is /fslhome/pjm43).

ls –al (-a shows hidden files). One of the files will be something like:

-rwxr-xr-x 1 pjm43 pjm43 1522 Dec 4 09:59 .bash\_profile

Edit the .bash\_profile using vi (or any other editor) by adding the following line to the bottom of the line (don’t forget to save the file):

export PATH=$PATH:/path/to/your/codedirectory

Now every time you open your shell, it will automatically know that programs are also in the code directory.

**Determining which processes are running (provides a list of running programs)**

top

ps

**Interrupting a program**

Ctrl+c

**Canceling a runaway program that can be stopped by Ctrl+c**

kill <PID>

The PID (process ID) can be gotten by running the command “ps” which show the process status.

**Count the number of words, line and characters in a file:**

wc somefilename.txt

if you want only to count the number of lines add –l, characters add –m, words add -w

**Count the number of a specific character in a file:**

grep –c ‘someword’ somefile.txt

grep –c ‘^>’ file.fasta would could the number of lines with leading >. Which would be the number of sequences in a fasta file.

-n adds the line number to the output

-B # prints the # of lines before each match

Say you wanted to identify poly-A motifs in the first 100 lines of your fasta file:

head –n 100 somefile.fasta | grep ‘aaaaaaaaaa’ –B 1 –n > output.txt

**Search and replace in unix (transliterate) – replace “a” with “A”**

tr ‘a’ ‘A’ < inputfile.fasta > outputfile.fasta

Say you wanted to change everything to all CAPITAL letters in your fasta:

tr ‘atcg’ ‘ATCG’ > newfile.fasta

*For pattern searches, use sed:*

sed ‘s/oldpattern/newpattern/ < somefile.txt > outputfile.txt

Another way to call this would be to use the cat command:

cat somefile.txt | head –n 100 | sed ‘s/oldpattern/newpattern/g’

the trailing “g” means that all patterns will be changed in the line. If you leave it off, only the first instance of the pattern in the line will be changed. This pipe is limiting the substitution to only the first 100 lines.

**Subselect only lines that contain a specific word:**

grep ‘someword’ somefile.txt > newfile.txt

-i ignore case (by default grep is case sensitive).

-v (--invert match) selected lines are those NOT matching any of the specified pattern

-w for isolated words (word is flanked by whitespaces)

**Sort a file:**

sort somefile.txt

-r for reverse

-n for numeric

**Commands can be piped together (|):**

grep ‘someword’ somefile.txt |sort > newfile.txt

**Cut out a portion of each line of a file**

cut –f 1-5 –d , <somecommadelimitedfile.cvs > newfilewithonly5column.cvs

#tab is made with cntrl-V <tab>