Advanced Unix Command Line

You have been provided with a tar file that contains some folder/files etc. for doing the below activities. Download the file and untar it (tar -xvf filename ; replace filename with the name of whatever you downloaded). It should create a folder called unix. Navigate to that folder. Then, navigate to the `file-analysis` folder. Most commands below assume you are inside the `file-analysis` folder. Note that you can also perform these activities on cLabs in Lab0.

wc

1. Ensure you are inside the file-analysis folder. wc prints word (-w), character (-m), line (-l), byte-count (-c) in a file. wc can accept zero (reads from standard input) or more inputs. Default output: lines, words, characters. The first command prints the lines, words and characters (in that order) of file bigfile. The second command prints only number of lines. Try the remaining commands also, they are straightforward.
    a. wc bigfile
    b. wc -l bigfile
    c. wc -w bigfile
    d. wc -m bigfile
    e. wc -c bigfile
2. We can use wc with multiple files (below we are using it across two files)
    a. wc /proc/cpuinfo /proc/meminfo

grep

1. Ensure you are inside the file-analysis folder. Grep: grep searches for PATTERNS in each FILE and prints each line that matches that pattern. Note that a line is not decided by full-stops in the para (like we humans do) but by a newline (\n). “-i” ignores the case. The first command will not show a sentence starting with “For”, whereas the second command will. The third command matches the exact pattern made of two words.
    a. grep “for” bigfile
    b. grep -i "for" bigfile
    c. grep -i "for a" bigfile
2. Regex Metacharacters (^ beginning of line; $ end of line; . match any single character)
    a. grep -i "^for" bigfile
    b. grep -i "cried.$" bigfile
    c. grep -i "h.s" bigfile
    d. grep -i "t..t" bigfile
3. More metacharacters: Suppose you want to use “or” operator i.e. “|”. The first command will fail, since it tries to match the same exact pattern. The second works, since by using \, you are escaping the | i.e you will interpret it as the “or” operator. However, this tends not to be clear when reading regular expressions. The best way around is the use of the

third command, use of -E (has to be capital) in which case you don't need to escape special characters. Use the man page of grep to know more.
   a. grep "cried|could" bigfile
   b. grep "cried\|could" bigfile
   c. grep -E "cried|could" bigfile
4. Quantifiers: * zero or more times; ? zero or one time; + one or more times; {n} exactly n times; {n,} at least n times; {,m} at most m times; {n,m} from n to m times
   a. grep -E "to*" bigfile
   b. grep -E "to?" bigfile
   c. grep -E "to+" bigfile
   d. grep -E "to{1}" bigfile
   e. grep -E "to{1,}" bigfile
5. Groups and Ranges: ( ) group patterns together; { } match a particular number of occurrences (saw this in 4.d and 4.e above); [ ] match any character from a range of characters. In the third command, the interpretation of “^” is not beginning of a line, rather since it appears within [], it means any letter except i. The fourth command, any letter in the alphabet range from A to Z (notice capitals, it will only match capital letters).
   a. grep -E "(fear)?less" bigfile
   b. grep -i "h[ia]s" bigfile
   c. grep 'h[^i]s' bigfile
   d. grep "[A-Z]" bigfile
6. Special characters. \s indicates white space. So, the first command will search for a three letter word that starts with h and ends in d; which has a space in front and at the end. \b matches any character that is not a letter or number without including itself in the match.
   a. grep -E "\sh.d\s" bigfile
   b. grep -E "\bh.d\b" bigfile
7. Grep is not restricted to just file search
   a. ls -l | grep file
8. More options of grep: -v negation, -r recursive search, -w whole word, -n show line number, -c count (count of lines, not number of occurrences). In the first command, every line that has the word “Wolf” is not printed! In the second command, only those lines that have “Wolf” are printed. In third command, we are using -r to search across all files in that folder (specified via *) i..e grep can work across files also, need not limit to one file.
   a. grep -v "Wolf" bigfile
   b. grep "Wolf" bigfile
   c. grep -r "Wolf" *
   d. grep -w "his" bigfile
   e. grep "his" bigfile
   f. grep -n "Wolf" bigfile
   g. grep -c "Wolf" bigfile

find

1. Locate files/directories with known patterns. Ensure you are in the parent of the student-files folder. The first command, -name tells to match the name of the given file (i.e. files ending in jpg) in the folder student-files. The second command is asking it to check only for files (-type f) ending in “.c” and be case insensitive in file names (-iname) in the current directory.
   a. find student-files/ -name "*.jpg"
   b. find . -type f -iname “*.c”
2. The first command lists all directories in the current directory (-type d)
   a. find . -type d
3. The first command finds files of size more than 60kbytes in current directory (use “ls -Rl .” to check the file sizes to verify if the command worked fine). The second command shows files with permission set to 644 in the current folder (permissions will be covered shortly). The last command, finds files starting with f in dir1 folder and then deletes the files within (be very careful when you execute this, since it will remove files)
   a. find . -type f -size +60k; ls -Rl
   b. find . -perm 644
   c. find dir1/ -name “f*” -delete

cut:

1. Ensure you are inside the file-analysis folder. cut: helps cut parts of a line by delimiter, byte position, and character (“-f” specifies field(s), “-b” specifies bytes(s), “-d” specifies a delimiter, --complement complements the selection and --output-delimiter specifies a different output delimiter string)
2. Below are some examples involving students.csv file. If you do cat students.csv, you will see various fields separated by comma (which is the delimiter). The first command specifies what delimiter to use (, in this case) and to output fields 1 and 3 (serial no and name). The second command is using -4 which means fields 1 to 4.
   a. cut students.csv -d ',' -f 1,3
   b. cut students.csv -d ',' -f -4
3. You can change the output delimiter as well. First command now separates the field by space when outputting.
   a. cut students.csv -d ',' -f 1,3 --output-delimiter=" "
4. When using complement, it outputs fields that are complement of 1,2,3 i..e 4,5,6,7,8
   a. cut students.csv -d ',' -f 1,2,3 --complement
5. Here are examples of -b. Note I am using echo and redirecting its output to cut. You don’t necessarily have to use a file
   a. echo "abcdefg" | cut -b 1,3,5
   b. echo "abcdefg" | cut -b 1-4

paste:

1. Helps merge lines of files horizontally. -d can be used to specify a delimiter when pasting. And -s pastes one file at a time in serial.
    a. paste fruits1 fruits2
    b. paste -d '_' fruits1 fruits2
    c. paste -s fruits1 fruits2