BIMM 185 Lab Report 1

Lihui Lu A99108553 4/11/2017

Unix Shell

During last week's lab, I was able to review unix shell usage as well as learn some new tricks while using the Unix shell. I have review the use of `-r` option as executing the command recursively. Some example usages of the `-r` flag:

```
rm -r directory #remove everything in the directory,including the directory
cp -r directory #copy everything in the directory, including directories
chmod -r directory #change the permission of everything in the directory,
#including the directory
```

Other than that, I also reviewed the usage of `ls`, listing the contents. One usage that I think would be useful when there are huge amount of files in one directory is `ls -lhrt` where `-l` will show detail information of each file; `-h` will show the size of files in human readable way; `-t` will sort the file base on modified time in descending order; `-r` will reverse the order and thus list the most recently modified file at the bottom.

I have also learned the usage of `-p` option for command `mkdir` which can create the whole directory hierarchy if some of the middle directories do not exists. This is a very convenient option so that I don't need to create directories in each level seperatly. What's more, combining this with the usage of setting alias in the environment, set `alias mkdir=mkdir -p` would easy the working process even more.

```
mkdir create/a/directory/hierarchy
output: mkdir: create/a/directory: No such file or directory

mkdir -p create/a/directory/hierarchy
#will create intermediate directories if not exist
```

Another useful trick that I learned last week is `cd -` which allows the user to go back to the

previous directory no matter which directory it was. It can be useful for people working on the terminal connecting a remote computing server and working among different directories. For example, I like to keep the all of my scripts in one place and the data to work with in another. Using `cd -` can used to switch between directories easily instead of typing out the full path repeatedly. In addition, `cd ~user` allows the user to access the home directory of another user directly.

```
cd - #go the the previous directory
cd ~user #go to the home directory of user
```

Loops can also be done through the unix shell. `;` is used as the line separator of the shell script. An example of unix shell for loop printing out the prefix of all the .txt files:

```
for txt in *.txt; do echo "${txt%.txt}";done

#for txt in *.txt -  will iterate through all the .txt file in the current dire
ctory.
#echo "${txt%.txt}" - will print out the prefix of the file,
#removing the trailing '.txt' in the file name
```

Data processing / Regular expression

We also learned some commands to process files. `sort` and `cut` are two basic commands that are used to work with files, simply modified them into formats that we want. Sometimes, only several columns in the files are necessary for the next program, possible solutions can be:

```
#Suppose that the input files contains 5 columns separated by ',',
#the required input for the next program only needs the 1st, 2nd and 5th column
,
#separated by tab.

cut -f 1,2,5 -d ',' --output-delimiter=$'\t' input > output

#-f cut by field
#-d ',' set delimiter to be ','
#--output-delimiter=$'\t' set delimiter in the output to be tab

#or we can also use awk
```

```
awk -F"," '{print $1"\t"$2"\t"$5}' input > output

#-F"," set delimiter to be ","
#$1, $2, $5 are the first, second, and the 5th column
```

`sort` can be used to sort the input in the desired order. Usage of sort is listed below:

```
sort input  #sort alphabetically
sort -f input #sort alphabetically, ignore case
sort -n input #sort by numeric value, this can be important since 10 will be be
fore 2 if sort alphabetically
sort -r input #reverse the sorting result
```

In addition, scripting languages can be used to process file in the command line. In class, we see examples of how to execute perl commands directly through command line. When processing data using perl, `-p` and `-n` were used as while loop while `-p` will print out each line as well. However, I didn't figure out how to execute python program directly through command line except writing a python script and call that script in the command line.

```
perl -pe #while loop and print out each line
perl -ne #while loop
```

The substitution usage of perl is very similar to substitution in vim. However, when using perl, we can specify to keep a backup of the original file through option `-i.bkp`.

```
In perl:
s/a/b/g will substitute every occurrence of a with b
s/a/b will substitute the first occurrence of a with b in each line

In vim:
:%s/a/b/g will substitute every occurrence of a with b
:%s/a/b will substitute the first occurrence of a with b in each line
```

When the size of input data is huge, more complex data process can be done through regular expression. Regular expression can be used to search for a specific format within the the input file. Some usages of the regular expression that I found useful are:

```
. matches any character except a newline
* wildcard, matches the preceding expression 0 or more times
+ matches the preceding expression 1 or more times
? will turn on the non-greedy mode and matches as few characters as possible wi
th the regular expression
#an example used in the manual is that while <.*> will match <a> b <c>, <.*?> w
ill match <a> only.
\S any character except space
[] will pick a character from anything listed inside
[0-9] single number between 0-9
[0-9\.-] anything between 0-9 or a . or a -
```

Scripting Challenges

Python codes for the scripting challenge 1 and 2 can also be found in the Week 1 folder through the GitHub Link: https://github.com/luna5124/BIMM185

Challenge 1

For challenge 1, the basic logic is that we want to read the file line by line, identify if the current line is a header line. If it is a header line, extract the information that we need and format it in the format we need. It it is not a header line, we just want to print out the sequence with no newline symbol until another header line is encountered.

Perl script that can be used for challenge 1:

```
perl -ne 'if (/>.*\|.*\|(.*)\|(\S+).*/) {print "\n$2-$1\t"} else {$_=~ m/\n/;pr
int $` }'
```

Python script for challenge 1:

```
import re
with open('TCDB.faa','r') as file:
    start = False
    for line in file:
        line = line[:-1]
        #check if the current line is in the format of a header line
        m = re.search('>.*\|.*\|(\S+)\|(\S+)',line)
```

```
        if m:
            if start:
                print()
            else:
                start = True

            #line = line.split('|')
            #print(line[3].split()[0] + '-' + line[2] + '\t',end='')

            #use regular expression
            print(m.group(2) + '-' + m.group(1) + '\t',end='')
        else:
            print(line,end='')
```

Challenge 2

For challenge 2, we also read the file line by line. This time, we will keep track of the reference protein(which is the protein in the first column), the max probability of this protein so far, the interacting protein which is giving the max probability, and the total count of proteins interacting with this reference protein.

A dictionary is used to store the information for each reference protein. The key of the dictionary is set to be the total count of interacting protein pairs for later sorting. The item for each key is a list of tuples where each tuple is `(reference protein, interacting protein with max probability, max probability)`

```
#Please note:
#Right is the protein in the first column(left)
#Left is the second column(right) in the input file.
#Since the usage of these two variables are consistent throughout the code,
#I decided to keep it as it was

with open('RS.txt','r') as file:

   #variable initialization
    right = ""
    left = ""
    my_max = 0
    count = 1
    protein = 0
    counts = {}
```

```
    for line in file:
        line = line[:-1].split('\t')
        if right == line[0]:
            count += 1
            if float(line[-1]) > my_max:
                #if the possibility of the read-in pair is higher
                #than the pairs with the same reference protein
                left = line[1]
                my_max = float(line[-1])
        else:
            if my_max != 0:
               #print(right, left, my_max)
                if count in counts:
                    counts[count].append((right, left, my_max))
                else:
                    counts[count]=[(right, left, my_max)]
            if protein == 2000:
                break
            #initialization when hit a new reference protein
            right = line[0]
            my_max = float(line[-1])
            left = line[1]
            count = 1
            protein += 1
    #in descending order of the number of interacting pairs
    for key in sorted(counts.keys(), reverse=True):
        for i in range(len(counts[key])):
            my_tuple = counts[key][i]
            #print my_tuple[0],'\t',my_tuple[1],'\t',float(my_tuple[2]),'\t', k
ey
            print my_tuple, '\t', key
```