Programming for Bioinformatics
BIOL 8803 B
September 28th, 2015

This week doesn't feature many new commands; we've covered most of the basic ones. Keep in mind that there are thousands of bioinformatics utilities that you can turn into commands. This week you will be writing your own command shell scripts.

Commands for today:

`sort` – sort some data

`uniq` – find unique lines in input

`tee` – Create a tee junction

`xargs` – Build and execute command lines from standard input

`seq` – Create a tee junction

command chaining operators

`if, elif, else` – Do different things depending on conditions

`for, while, until, select` – Loop construct

`getopts` – Get arguments from the command line

`case` – Look at what possibilities you have for a variables value

**Exercises**

1.) More piping with `sort` and `uniq`

Create the following file:

```
perl -e 'foreach(1..100){print $_."\n".($_ / 2)."\n"}' > uniq.txt
```

a.) Sort the file numerically, and output it to another file without using '>'.

b.) Pipe the result of the `sort` to `uniq`. What happened?

c.) Pipe the result of the `sort` to `uniq`, and discard all lines that **appear more than once**. i.e., I don't want the lines (e.g. 1, 2, etc) which occur more than once.

d.) Pipe the result of the `sort` to `uniq` and **count** the number of times each number appears

2.) Command understanding

For next week, take apart the following commands and write out what they do. You should be able to explain step-by-step what each command does.

a.) `ls –ls * | sort –k 6n,6n | tail -5`

b.) `ps a | awk '{print $1}' | xargs –I one echo kill one`

c.) `cat a_file.txt | xargs –I one cat one >> all_files.txt`

3.) So you think you can `sort`?

a.) Take the first 4 columns of the original ref.bed file of question 4 using `awk` and save it as file w5q5.bed

b.) `sort` the file first by chr (column 1) followed by start (column 2), stop (column 3) and finally the accession (column 4) in that order. This is not going to be super easy. Make sure the chromosomes are sorted in ascending order, i.e., chr1, chr2, chr3, … and not as chr1, chr10, chr11, …

c.) May be that wasn't that bad. Let's make our lives more interesting. Add header to w5q5.bed by using only `echo && cat`. The header is simply "`chr\tstart\tstop\taccession`". To be clear – you are not using any text editors or changing this by hand. You need to use commands to do this. Name this file w5q5c.bed

d.) Now let's sort the file again, but this time, make sure the header stays put. I.e., your first line should not change from the header in the sorting process. Save it as a new file w5q5d.bed

NOTE: Your command should be a **general** one, not something that would work exclusively for this file. You should be able to take that command and do the same thing with any file with similar requirement.

e.) Get our best friend awk to add an extra column to the file w5q5c.bed, name it length, and as the name suggest, the column will contain the length of the gene (stop – start + 1). Also compute the average length of all gene and have it as the last line of the file. Average = <whatever number comes up>

Again – all this using awk, no editors or anything else. Pure awk. Hardcore. Output file: w5q5e.bed

f.) You guessed it, you are going to sort this file! Yay! This time make sure the header stays put as well as the footer (the last line containing the average). Hint: If it works for two, it will work for three.

4.) Passing arguments with `xargs`

a.) List the files in the directory, and pass them to the `head` command using `xargs` and the `-I` flag

b.) Make three files, `one.fas`, `two.fas` and `three.fas`. Use `sed` and `xargs` to change the extension to `.fna`

c.) Get the first two columns of the UCSC gene file using `cut`

d.) Get the first two columns of the UCSC gene file using `xargs` and `echo`

5.) Multiple arguments with `xargs`

a.) Print the numbers 1 to 12

b.) Feed these to xargs in multiple of 3 and print them as:

First number: 1; second number 2; third number 3;
First number: 4; second number 5; third number 6;
First number: 7; second number 8; third number 9;
First number: 10; second number 11; third number 12;

6.) `seq` along

a.) Print the sequence of numbers from 1 to 50

b.) Repeat (a) in reverse order

c.) Print 1 to 50 in steps of 5

d.) Print 1 to 10 followed by 9 to 1 using chained seq commands.

7.) It's not a game anymore!

a.) Extract kgXref file.

b.) What is the information that these two files (knownGene and kgXref) contain? The column names can be fetched from the accompanying SQL files.

c.) Replace all the tab characters (represented by `\t` in regex) by a period ( **.** ) using `sed`. Hint: a single `sed` replacement will not replace all the tabs. Why?

d.) `sort` the two files individually by the first column (kgID) and store the results as two new files

e.) Extract the columns: kgID, gene symbol and refseq accession from the sorted kgXref file using `awk` and store it in a new file – say colsKgXref.txt

f.) Extract the columns: chromosome, strand, transcription start and end from the sorted knownGene file using `awk` and store it in a new file – say colsKnownGene.txt

g.) Merge the two files by columns using `paste`

h.) Explain what you accomplished by the above steps

**Basic shell scripting**

8.) Copy the exact content from the following slides and run it yourself. I want each one of you to write this down yourself and run it once to get the confidence on how these scripts work. They are not hard but will continue to daunt you unless you do it once. Of course, feel free to keep changing lines

a.) Slide 39

b.) Slide 41

c.) Slide 42

d.) Slide 52

e.) Slide 60

f.) Slide 63

9.) Write shell scripts for the following:

a.) Write a shell loop that adds up all the numbers from 1-100

b.) Write a shell loop that prints each letter in the word “Hello” separately. This can be done using a for loop iterating through the letters H e l l o. This is a 5 line script, don’t make it harder than that.

c.) Have variable x increase from 0 to 100 and variable y decrease from 100 to 0 by 1 unit at a time and print “x and y are equal” when they are equal

d.) Repeat the above but this time print the difference between x and y if they are not equal

e.) Write the a getopts block with 3 options including 1 that requires an argument

f.) Write a script utilizing `case` that asks the user the day of the week and prints the day number based on user input

10.) Mini challenge.
We are going to carry out an exercise for fun but it would enforce shell concepts and will help you prep up for next week. So take this with a heart of lion!

To make things easier for you, I have provided you with stepwise instructions on how to proceed. You will add one element each time making the loop a little more complicated as compared to the last time.

a.) Write a `for` loop to create 10 files by the name seq1.fasta, seq2.fasta, seq3.fasta and so on. You can create a file using the touch command.

b.) Modify this loop to now do two things:

i) Delete the seq1.fasta, seq2.fasta, etc. files if they exist.

ii) Create a new seq1.fasta, seq2.fasta, etc. file with fasta description line in it. The fasta description line needs to be like this: >seq1 for seq1.fasta file, >seq2 for seq2.fasta file and so on

c.) Add another element to this loop: the loop now also adds a random DNA sequence along with the fasta description line. The following command will give you a random DNA string of 50*10 letters.

```
cat /dev/urandom | tr -dc 'ACGT' | fold -w 50 | head
```

d.) Let’s try nesting the loops. Nesting loops is really having a loop within a loop. Instead of creating 10 single sequence fasta files, we will create 10 multiple sequence fasta files (a.k.a. multi-fasta file) with 8 sequences in each. The fasta descriptor for file1 will go like these: >seq1_1, >seq1_2, >seq1_3, etc. The fasta descriptor for file2 will go like these: >seq2_1, >seq2_2, >seq2_3, etc.

e.) Now add getopts to the beginning of the script. I only want to take two options:

i) Option ‘n’ => will take number as an input. The argument described the number of files to be created (which till this point was 10)
ii) Option ‘m’ => will take number as an input. The argument described the number of sequences to be added in each file (which was 8)
iii) Option ‘v’ => verbose mode, i.e. if I want the script to print out every single action it is doing. This is a flag so no input is expected with it.

**Deliverables**

- Commands and explanations in a word or PDF file
- Code: for question 10)
    - week6.sh