Programming for Bioinformatics

BIOL 7200

October 3rd, 2016

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 – get a sequence of numbers
* Command chaining operators
* if, elif, else – do different things depending on conditions
* for, while, until, select – loop constructs
* getopts – get arguments from the command line
* case – look at what possibilities you have for a variable’s value

**Instructions for submission:**

* You will be required to submit a **brief** **solution sheet** and a **shell script** (see question 10)
* **This exercise has an associated quiz that can be found on T-Square – due October 17!**

**Exercises**

1. More piping with sort and uniq

Create the following file:

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

* 1. sort the file numerically and output it to another file without using ‘>’

“sort -o sort1.txt uniq.txt”

* 1. Pipe the result of the sort to uniq. What happened?

“sort -n uniq.txt | uniq”

-n does numeric sort.

uniq removes multiple occurrence of the numbers.

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

sort -n uniq.txt | uniq -u

option -u represents only unique lines.

* 1. Pipe the result of the sort to uniq and count the number of times each number appears

sort -n uniq.txt | uniq -c

option -c prefixes lines by the number of occurrences

1. 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.

* 1. ls -ls \* | sort -k 6n,6n | tail -5

ls – ls \* : This lists all the files/folders and sub-files/folders(\*) in a list format(-l) with their size in blocks (-s).

sort –k 6n,6n : sorts according to key(-k), which is in this case starting and ending of the 6th column(6n,6n). If there is no 6th column, sort sorts the lines alphabetically from the start of the line(default)

tail -5 : displays the last 5 lines of the input.

* 1. ps a | awk ‘{print $1}’ | xargs -I one echo kill one

ps a : Displays all running processes with a terminal

awk ‘{print $1}’ : the 1st column of the input and displays through the standard output.

xargs -I one echo kill one : -I replaces string; replaces standard input with the variable ‘one’ and executes ‘echo kill one’; which displays the first column with word kill.

In other words, ps displays all processes, piped to awk, which selects first column i.e. PID; which is again piped to xargs, which receives first data in first column in variable 'one' and fires echo, which prints 'kill <value of one>' for every PID.

* 1. cat a\_file.txt | xargs -I one cat one >> all\_files.txt

cat a\_file.txt : displays the contents of a file ‘a\_file.txt’

xargs –I one cat one >> all\_files.txt : xargs takes input and assigns it to variable ‘one’ 1 line each at a time and then fires cat which will concatenate the contents of the file whos filename is stored in to variable ‘one’ to all\_files.txt.

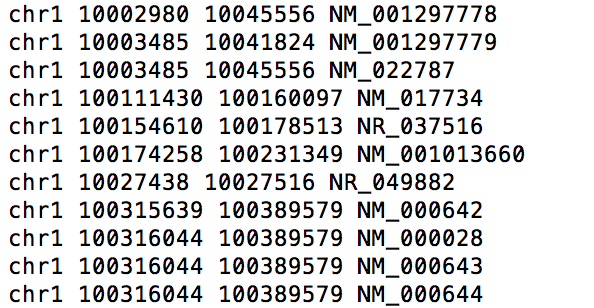
In other words, this would only work if file a\_file.txt is having a list of filenames, each in a separate line, which are contained in the present working directory. Cat will send all those filenames to standard output, which is piped to xargs, which assigns each filename each time to variable ‘one’ by switch –I and fires cat command afterwards which concatenates the content of the file to all\_files.txt. this will conatin contents of all the files in present working directory.

1. So you think you can sort?
   1. Take the first 4 columns of the original ref.bed (from week 5) using awk and save it to as the file **w6q3.bed**

“awk '{print $1,$2,$3,$4}' ref.bed>w6q3.bed”

* 1. 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, …

sort -k1.4 -k 2 -k 3 –k 4 w6q3.bed > temp.bed



* 1. Maybe that wasn’t too bad. Let’s make our lives more interesting. Add a header to w6q3.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 as **w6q3c.bed**

(echo -e 'chr\tstart\tstop\taccession'&&cat w6q3.bed) > w6q3c.bed

* 1. Now let’s sort the new file, 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 **w6q3d.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 the do the same thing to any file with a similar requirement.

“awk ‘NR==1;NR>1{print $0 | “sort –V”} w6q3c.bed > w6q3d.bed”

* 1. Get our best friend awk to add an extra column to the file w6q3d.bed, name it “length”, and as the name suggests, the column will contain the length of the gene (stop - start + 1). Also compute the average length of all genes and have it as the last length of the file: Average = <whatever number comes up>

Again – all of this should be done using awk, not editors or anything else. Pure awk; hardcore. Output file: **w6q3e.bed**

Awk ‘BEGIN{counter=0;print “chrom\tstart\tstop\taccession\tlength”} NR>1{OFS=”\t”; $5=($3-$2+1); counter=counter+$5; sum=sum+$5; print $0} END{print sum/NR}’ w6q3d.bed > w6q3e.bed

* 1. You guessed it, you’re 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.

Head –n 1 w6q3e.bed && tail –n +2 w6q3e.bed | sort && tail –n 1 w6q3e.bed

1. Passing arguments with xargs
   1. List the files in the directory and pass them to the head command using xargs and the -I flag

Ls -1 : lists all the filenames in a separate line; xargs then sends each filename to head to display first 10 lines of each file.

“ls | xargs –I list head list

* 1. Make three files: one.fas, two.fas and three.fas. Use sed and xargs to change the extensions to .fna

ls all .fna files; sed then chops off their extensions; xargs then renames each of them with

extension fna

“ls \*.fas|sed 's/fas$//'|xargs -I NAME mv NAMEfas NAMEfna”

* 1. Get the first two columns of the UCSC gene file (from week 5 – ref.bed) using cut

“cut -f1,2 –s ref.bed”

-s stands for selection of only delimited files; -f<value> selects the field nos.

* 1. Get the first two columns of the UCSC gene file using xargs and echo

“awk ‘{print $1,$2}’ ref.bed | xargs –I file echo file

1. Multiple arguments with xargs
   1. Print the numbers 1 to 12

“echo {1..12}”

* 1. Feed these numbers (from 5a) to xargs in multiples 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

The output should look like this:

1 2 3

4 5 6

7 8 9

10 11 12

“echo {1..12} | xargs –n 3”

1. seq along
   1. Print the sequence of numbers from 1 to 50

“Seq 1 50”

* 1. Repeat (a) in reverse order

“seq 50 -1 1”

* 1. Print 1 to 50 in steps of 5

“seq 1 50|xargs –n 5” or if we need to print in increments of 5 use “seq 1 5 50”

* 1. Print 1 to 10, followed by 9 to 1 using chained seq commands

“seq 1 10 ; seq 9 -1 1”

1. It’s not a game anymore!
   1. Extract the kgXref file from week 3.

“gunzip kgXref.txt.gz”

* 1. What is the information contained in the knownGene and kgXref files (from week 3)? The column names can be fetched from the accompanying SQL files.

knownGene.sql:

name, chrom, strand, txStart, txEnd, cdsStart, cdsEnd, exonCount, exonStarts, exonEnds, proteinID alignID

* protein coding genes based on proteins from UniProtKB and their corresponding mRNA from GeneBank.

kgXref.sql:

kgID, mRNA, spID, spDisplayID, geneSymbol, refseq, protAcc, description, rfamAcc, tRnaName

* links together a known gene ID and mRNA, UniProtKB, RefSeq and NCBI accession ID’s
  1. Replace all the tab characters (represented by \t in regex) with a period (.) using sed. Hint: a single sed replacement will not replace all the tabs. Why?

“sed ‘s/\t/\./g’ kgXref.txt”

“sed ‘s/\t/\./g’ knownGene.txt”

/g globally replaces.

* 1. sort the two files individually by the first column (kgID) and store then results as two new files

“sort -k1,1n knownGene.txt > sortgene.txt”

“sort -k1,1n kgXref.txt > sortXref.txt”

* 1. Extract the columns kgID, gene symbol and RefSeq accession from the sorted kgXref file using awk and store it in a new file – **colsKgXref.txt**

“awk ‘{print $1, $5, $6}’ sortXref.txt > colsKgXref.txt”

* 1. Extract the columns chromosome, strand, transcription start and transcription end from the sorted knownGene file using awk and store it in a new file – **colsKnownGene.txt**

Awk ‘{print $2, $3, $4, $5}’ sortgene.txt > colsKnownGene.txt

* 1. Merge the two files by columns using paste

“paste colsKnownGene.txt colsKgXref.txt > merge.txt”

* 1. Explain what you accomplished in the above steps

The new file obtained after ‘paste’ contains information of a known gene with its kgID, gene symbol and refseq accession ID along with information on chromosome number, strand, transcription start and end site. This is sorted in ascending alphanumeric order of chromosome number and kgID.

**Basic shell scripting**

1. Copy the exact content from the following slides and run them yourself. I want each one of you to write this down yourself and run it once to gain confidence on how these scripts work. They are not hard, but you will continue to be daunted by this unless you do it once. Of course, feel free to keep changing lines.
   1. Slide 39
   2. Slide 41
   3. Slide 42
   4. Slide 52
   5. Slide 57
   6. Slide 60
   7. Slide 63
2. Write shell scripts for the following:
   1. Write a shell loop that adds up all the numbers from 1-100

**#!/bin/bash**

**echo "adding numbers from 1-100"**

**n=100**

**i=1**

**sum=0**

**while [ $i -le $n ]**

**do**

**sum=$((sum+$i)) #calculates sum of digits**

**i=$(($i+1))**

**done**

**echo "the sum is $sum"**

* 1. Write a shell lop 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.

**#!/bin/bash**

**s=hello**

**for i in $(seq 1 ${#s})**

**do**

**echo ${s:i-1:1}**

**done**

* 1. 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 the variables are equal

**#!/bin/bash**

**x=0**

**y=100**

**while [ $x -ne 101 ]**

**do**

**if [ $x -eq $y ]**

**then**

**echo “x is equal to y”**

**echo $x**

**echo $y**

**fi**

**x=$[$x+1]**

**y=$[$y-1]**

**done**

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

**#!/bin/bash**

**x=0**

**y=100**

**while [ $x -ne 101 ]**

**do**

**if [ $x -ne $y ]**

**then**

**val=$(($x-$y))**

**echo $val**

**fi**

**x=$[$x+1]**

**y=$[$y-1]**

**done**

* 1. Write a getopts block with 3 options including 1 that requires an argument

**#!/bin/bash**

**while getopts ":abc:" opt;**

**do**

**case $opt in**

**a)**

**echo "-a is triggered!";;**

**b)**

**echo "-b is triggered!";;**

**c)**

**echo "-c is triggered!";;**

**\?)**

**echo "invalid option";;**

**:)**

**echo "option -$OPTARG requires an argument.";;**

**esac**

**done**

* 1. Write a script utilizing case that asks the user the day of the week and prints the day number based on user input

**#!/bin/bash**

**Day=$1**

**case $Day in**

**monday)**

**echo "1";;**

**tuesday)**

**echo "2";;**

**wednesday)**

**echo "3";;**

**thursday)**

**echo "4";;**

**friday)**

**echo "5";;**

**saturday)**

**echo "6";;**

**sunday)**

**echo "7";;**

**\*)**

**echo "invalid please give the day of the week";;**

**esac**

1. Mini challenge

We are going to carry out an exercise for fun, but it will enforce shell concepts and will help you prepare for next week.

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

* 1. 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.

**#!/bin/bash**

**i=0**

**for i in {1..10}**

**do**

**touch seq${i}.fasta**

**done**

* 1. Modify this loop to now do two more things:
     1. Delete the seq1.fasta, seq2.fasta, seq3.fasta, etc. if they exist
     2. Create new seq1.fasta, seq2.fasta, seq3.fasta, etc. files with a FASTA description line in it. The FASTA description lines needs to be like this: >seq1 for seq1.fasta, >seq2 for seq2.fasta, >seq3 for seq3.fasta, etc.

**for i in {1..10}**

**do**

**if [ -e seq${i}.fasta ] #-e means the file exists**

**then**

**rm seq${i}.fasta**

**# echo "file exists and deleted"**

**fi**

**touch seq${i}.fasta**

**echo ">seq${i}" >> seq${i}.fasta # >> to append the required line**

**done**

* 1. 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 50x10 letters.

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

**for i in {1..10}**

**do**

**if [ -e seq${i}.fasta ] #-e means the file exists**

**then**

**rm seq${i}.fasta**

**# echo "file exists and deleted"**

**fi**

**touch seq${i}.fasta**

**echo ">seq${i}" >> seq${i}.fasta # >> to append the required line**

**cat /dev/urandom | tr -dc 'ACGT' | fold -w 50 | head >> seq${i}.fasta**

**done**

* 1. Let’s try nesting the loops. Nesting loops is really having a loop within a loop. Instead of creating 10 single sequence files, we will create 10 multiple sequence FASTA files (aka, 10 multi-FASTA files) with 8 sequences in each. The FASTA descriptors for file 1 will go like this: >seq1\_1, >seq1\_2, >seq1\_3, … ; the FASTA descriptor for file2 will go like this: >seq2\_1, >seq2\_2, >seq2\_3, … ; etc.

**for i in {1..10}**

**do**

**if [ -e seq${i}.fasta ] #-e means the file exists**

**then**

**rm seq${i}.fasta**

**# echo "file exists and deleted"**

**fi**

**touch seq${i}.fasta**

**for j in {1..8}**

**do**

**echo ">seq${i}\_${j}" >> seq${i}.fasta # >> to append the required line**

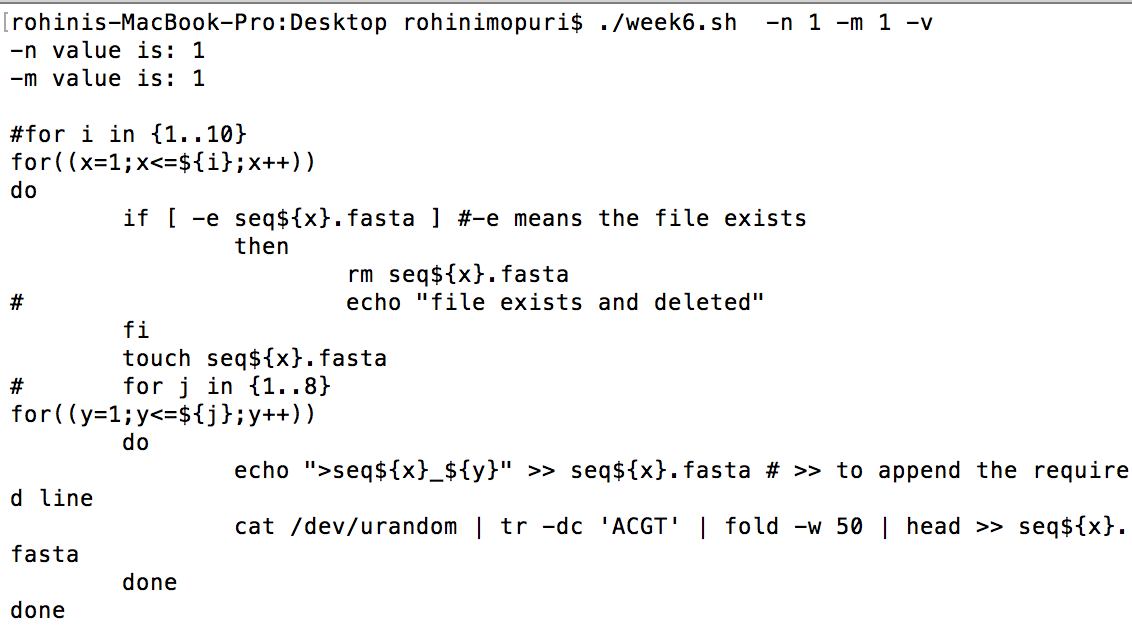
**cat /dev/urandom | tr -dc 'ACGT' | fold -w 50 | head >> seq${i}.fasta**

**done**

**done**



* 1. Now add getopts to the beginning of the script. I only want to take three options:
     1. Option ‘n’ => will take a number as an input. This argument describes the number of files to be created (which until now was 10)
     2. Option ‘m’ => will take a number as an input. This argument describes the number of sequence to be added in each file (which until now was 8)
     3. Option ‘v’ => verbose mode, i.e., I want the script to print out every single action it is doing. This is a flag, so no input is expected with it



**Write the *FINAL* version of the commands for question 10 in one shell script. Name this script as week6.sh and submit this script on T-square.**