Programming for Bioinformatics
BIOL 8803 B
September 28th, 2015

Exercises

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

Create the following file:

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

```
Here it prints the numbers from 1 to 100 with their halves.
```

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

tannishtha@tsom3:~$ sort -n uniq.txt >> uniqSorted.txt

-n is used for sorting numerically.

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

tannishtha@tsom3:~$ uniq uniqSorted.txt

All adjacent matching lines are removed.

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.

tannishtha@tsom3:~$ uniq -u uniqSorted.txt

0.5
1.5
2.5
3.5

-u is to print only the unique lines ,that is, the numbers which do not have any repetitions.

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

tannishtha@tsom3:~$ uniq -c uniqSorted.txt

1 0.5
2 1
1 1.5
2 2
1 2.5

-c is is used to count the number of occurrences of each number.

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`

ls -ls * - Prints the details and allocated size of each file and also of the files in the subdirectories present in the current directory.

Sort -k 6n,6n – This sorts the printed files in numeric order by the 6th column ie by the size of the files.

Tail -5 - this will print the last 5 lines of the sorted files got from the previous pipe.

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

ps a – print all the processes in the tty and pts terminals

awk '{print $1}' – This will print the 1st column ie the PIDs of the processes got from the previous pipe.

Then xargs will replace all the PIDs (-I is used for string replacement) with "kill PID"

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

This takes the file names written in a_file.txt and concatenates their contents to 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

tannishtha@tsom3:~$ awk '{print $1,$2,$3,$4}' ref.bed > 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, …

Sorting by chromosome - tannishtha@tsom3:~$ sort -V -k1,1 w5q5.bed

Sorting by start - tannishtha@tsom3:~$ sort -n -k2,2 w5q5.bed

Sorting by stop - tannishtha@tsom3:~$ sort -n -k3,3 w5q5.bed

Sorting by Accession number - tannishtha@tsom3:~$ sort -V -k4,4 w5q5.bed

Sorting all of them together - tannishtha@tsom3:~$ sort -V -k1,1 -k2n,2 -k3n,3 -Vk4,4 w5q5.bed

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

tannishtha@tsom3:~$ (echo -e "chr\tstart\tstop\taccession" && cat w5q5.bed) > 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

tannishtha@tsom3:~$ (head -1 w5q5c.bed && tail -n +2 w5q5c.bed | sort -V -k1,1 -k2n,2 -k3n,3 -k4,4) > w5q5d.bed

Here the header is printed first with the head command and then the rest of the lines are taken by tail command and sorted as in (b).

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>Output file: w5q5e.bed

tannishtha@tsom3:~$ awk 'BEGIN{total_length=0} {if(FNR==1) {OFS="\t";print $0,"length"} else{len=$3-$2+1; total_length+=len; print $0,len}} END{Average=total_length/(NR-1); print "Average =",Average}' w5q5c.bed > w5q5e.bed

Here in the header(FNR=1), length column is added. Then for each of the next lines, length of the gene is calculated and printed. Also the total length of all the genes are calculated. At the end of the file,

average length is calculated by dividing total length with NR-1 as the first line is the header. Average found is 55397.6.

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.

tannishtha@tsom3:~$ (head -1 w5q5e.bed && tail -n +2 w5q5e.bed | head -n -1 | sort -V -k1,1 -k2n,2 -k3n,3 -k4,4 && tail -1 w5q5e.bed) > w5q5f.bed

Here the first head command prints the first line of the file (the one with header) and then tail and head are used to select the lines containing the chromosome details except the last line and sort them as in (b) and then the last line is picked up by the tail command.

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

tannishtha@tsom3:~$ ls | xargs -I one head one > head.txt

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

tannishtha@tsom3:~$ nano one.fas

tannishtha@tsom3:~$ nano two.fas

tannishtha@tsom3:~$ nano three.fas

tannishtha@tsom3:~$ ls *.fas | sed 'p;s/\.fas/\.fna/' | xargs -n2 mv

Here ls lists the files with .fas extention. Then sed command is used to print the name of the original file (p flag) and also the name of the modified file. Now xargs will take 2 filenames and move them with the same name.

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

tannishtha@tsom3:~$ cut -f1,2 ref.bed

-f option is used to select the required fields.

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

tannishtha@tsom3:~$ cat ref.bed | xargs -L1 sh -c 'echo $0 $1'

-L will extract one line from the file and indicate the shell to perform function of displaying the first two columns

5.) Multiple arguments with `xargs`

a.) Print the numbers 1 to 12

tannishtha@tsom3:~$ echo {1..12} | xargs

1 2 3 4 5 6 7 8 9 10 11 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;

tannishtha@tsom3:~$ echo {1..12} | xargs -n3 sh -c 'echo "first number:$0; second number:$1; third number:$2"'
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
Here the -n option with xargs is used to indicate the maximum number of arguments per command line.

6.) `seq` along

a.) Print the sequence of numbers from 1 to 50
tannishtha@tsom3:~$ seq 1 50
b.) Repeat (a) in reverse order
tannishtha@tsom3:~$ seq 50 -1 1
Here the first number is the start, second is the increment (here by -1) and third is the last number.
c)Print 1 to 50 in steps of 5
tannishtha@tsom3:~$ seq 1 5 50
Here 5 means increment by 5.
d)Print 1 to 10 followed by 9 to 1 using chained seq commands.
tannishtha@tsom3:~$ seq 1 10 ; seq 9 -1 1
Here ; is used to chain the seq commands.

7.) It's not a game anymore!

a.) Extract kgXref file.
tannishtha@tsom3:~$ gunzip kgXref.txt.gz
As the file was already downloaded from a previous exercise, I just extracted it here with gunzip command.
b.) What is the information that these two files (knownGene and kgXref) contain? The column names can be fetched from the accompanying SQL files.
The knownGene file contains information about the coding regions on the human chromosomes with the fields giving information about the transcription start and end, the coding sequence start and end, the exon counts and the exon start and end.
The kgXref file contains information about the mapping between mRNA and refSeq genes.
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?
tannishtha@tsom3:~$ sed -r 's/\t{1,}/./g' kgXref.txt | less
As some of the columns are separated by one or more tabs, sed search string has to take that into account otherwise two tabs will be replaced by two periods.
d) `sort` the two files individually by the first column (kgID) and store the results as two new files
Sorting the kgXref.txt file by 1st column :
tannishtha@tsom3:~$ sort -k1,1 kgXref.txt > kgXrefSorted.txt

Sorting the knownGene.txt file by 1st column :

tannishtha@tsom3:~$ sort -k1,1 knownGene.txt > knownGeneSorted.txt

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

tannishtha@tsom3:~$ awk 'BEGIN{FS="\t"}{OFS="\t";print $1,$5,$6}' kgXrefSorted.txt > colsKgXref.txt

The kgID, gene symbol and refseq accession are the columns 1,5 and 6 respectively.

As the default field separator for awk is space, to take into account that the field separator in the file is a tab, FS has to be set to tab otherwise it doesn't recognise an empty field in a column.

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

tannishtha@tsom3:~$ awk 'BEGIN{FS="\t"}{OFS="\t";print $2,$3,$4,$5}' knownGeneSorted.txt > colsKnownGene.txt

chromosome, strand, transcription start and end are the columns 2,3,4,5 respectively.

g.) Merge the two files by columns using paste

tannishtha@tsom3:~$ paste colsKgXref.txt colsKnownGene.txt > XrefKnownGeneMerged.txt

h.) Explain what you accomplished by the above steps

After pasting, the file contains the list of known gene IDs with gene symbol , refSeq ID, chromosome, strand, transcription start and end sorted by the known gene IDs. Thus it gives us the information about the known genes with the refSeq IDs and their location with trancription details.

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

```
tannishtha@tsom3:~$ ./arithm.sh
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
a is not equal to b
```

b.) Slide 41

```
tannishtha@tsom3:~$ ./relation.sh
10 -eq 20 : a is not equal to b
10 -ne 20 : a is not equal to b
10 -gt 20 : a is not greater than b
10 -lt 20 : a is less than b
10 -ge 20 : a is not greater than or equal to b
10 -le 20 : a is less than or equal to b
```

c)Slide 42

```
tannishtha@tsom3:~$ ./boolean.sh
10 != 20 : a is not equal to b
10 -lt 100 -a 20 -gt 15 : returns true
10 -lt 100 -o 20 -gt 100 : returns true
      10  -lt 5 -o 20 -gt 100  : returns false
```

d)Slide 52

```
tannishtha@tsom3:~$ ./control.sh
a is less than b
```

e.) Slide 60

```
tannishtha@tsom3:~$ ./while.sh
0
1
2
3
4
5
6
7
8
9
```

f.) Slide 63

```
tannishtha@tsom3:~$ ./until.sh
0
1
2
3
4
5
6
7
8
9
10
```

9.) Write shell scripts for the following:

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

```
tannishtha@tsom3:~$ cat add.sh
#!/bin/bash
a=1
sum=0
while [ $a -le 100 ]
```

```
do
        sum=`expr $sum + $a`
        a=`expr $a + 1`
done
echo "Sum of numbers from 1 to 100 is $sum"
```

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.

```
tannishtha@tsom3:~$ cat hello.sh
#!/bin/bash
a="hello"
for (( i=0; i<${#a}; i++ )) #length of the string is given by ${#a}
do
        echo -n -e ${a:$i:1}"\t" #${a:$i:1}  #extracts 1 character from ith position of a
done
```

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

```
#!/bin/bash
for (( x=0, y=100; x <= 100, y >= 0; x++, y-- ))
do
    echo -n "$x $y"
    if [ $x -eq $y ]
    then
        echo "  x and y are equal"
    else
        echo -e "\t"
    fi
done
```

Output :

```
0 100
1 99
2 98
49 51
50 50  x and y are equal
51 9
```

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

```
#!/bin/bash
for (( x=0, y=100; x <= 100, y >= 0; x++, y-- ))
do
    echo -n "$x $y"
    if [ $x -eq $y ]
    then
        echo -e "\t"
```

```
    else
        a=`expr $x - $y`
        echo "  Difference between $x and $y is $a"
    fi
done
```

Output :

```
0 100  Difference between 0 and 100 is -100
1 99  Difference between 1 and 99 is -98
2 98  Difference between 2 and 98 is -96
49 51  Difference between 49 and 51 is -2
50 50
51 49  Difference between 51 and 49 is 2
```

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

```
#!/bin/bash
while getopts "ab:c" opt
do
   case $opt in
      a)echo "-a was triggered" >&2
        ;;
      b) echo "-b was trigerred, Parameter: $OPTARG" >&2
         ;;
      c) echo "-c was triggered " >&2
        ;;
      \?)echo "Invalid option: -$OPTARG" >&2
         exit 1
        ;;
      :) echo "Option -$OPTARG requires an argument" >&2
        exit 1
    esac
done
```

f.) 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
shopt -s extglob  #case can take regular expression in its options
while true
do
   read -p "Enter the day of the week:" day
   case $day in
      [Mm]on?(day) ) echo " It is day 1 of the week"; break;;
      [Tt]ue?(sday)) echo " It is day 2 of the week"; break;;
      [Ww]ed?(nesday)) echo " It is day 3 of the week"; break;;
      [Tt]hur?(sday)) echo " It is day 4 of the week"; break;;
```

```
        [Ff]ri?(day)) echo " It is day 5 of the week"; break;;
        [Ss]at?(urday)) echo " It is day 6 of the week"; break;;
        [Ss]un?(day)) echo " It is day 7 of the week"; break;;
        *) echo " Invalid option"; break;;
    esac
done
```

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!

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.

```
#!/bin/bash
for i in {1..10}
do
   touch seq$i.fasta
done
```

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

```
#!/bin/bash
for i in {1..10}
do
   a="seq$i.fasta"
   if [ -f $a ]
   then
     rm $a
   fi
   echo ">seq$i" > $a
done
```

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
#!/bin/bash

for i in {1..10}
do
   a="seq$i.fasta"
   if [ -f $a ]
   then
```

```
    rm $a
  fi
  echo ">seq$i" > $a
  dna=$(cat /dev/urandom | tr -dc 'ACGT' | fold -w 50 | head)
  echo "$dna" >> $a
done
```

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.

```
#!/bin/bash
for i in {1..10}
do
   a="seq$i.fasta"
   if [ -f $a ]
   then
      rm $a
    fi
     for j in {1..8}
     do
        echo ">seq$i""_$j" >> $a
        dna=$(cat /dev/urandom | tr -dc 'ACGT' | fold -w 50 | head)
        echo "$dna" >> $a
     done
done
```

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.

This is in the script file named week6.sh