1 Solutions to Unix for Bioinformatics

1.1 Introduction

No questions in this section.

1.2 Basic Unix

1. `ls -al`
2. There are 4 files in the directory (and 2 subdirectories). You can use `ls -l` to look inside the directory. This will show you which of the contents are files and which are directories. Don't forget to also include the `-a` option to show any hidden files:

```
ls -la Pfalciparum
```

3. Malaria.fa is the largest file. You can add the `-h` option to the command above to make the size of the files more readable.

4. `cd Pfalciparum`

5. There is one file in the fasta directory (hint: it's a hidden file!).

6. `cp Pfalciparum.bed annotation`

7. `mv *.fa fasta`

8. 4 files.

9. There are 6 GFFs in the unix directory. To search from the Unix directory, you can either use `cd` to move up to the directory, or you can specify the path in the `find` command. This can either be the absolute path, which you can get from `pwd`, or you can use the relative path, like so:

```
find ../../.. -name *.gff
```

10. There are 7 fasta files in the unix directory. Note that fasta files normally end with `.fa` OR `.fasta`, so wee need to make sure we look for both of these, by adding a wildcard after `fa`:

```
find ../../.. -name *.fa*
```

1.3 Looking inside files

1. `head -n 500 Styphi.gff > Styphi.500.gff`

2. There are 6213 lines in the file. Use the `-l` option:

```
wc -l Pfalciparum.bed
```

3. `sort -k 1 -k 2 -n Pfalciparum.bed`

4. Here is one way to do this: First use `awk` to get the first column of the file. Sort this and then use the `-c` option for `uniq` to count how many entries each chromosome has:

```
awk '{ print $1 }' Pfalciparum.bed | sort | uniq -c
```

The expected output would be:

```
190 01
264 02
287 03
292 04
357 05
373 06
395 07
374 08
425 09
452 10
553 11
621 12
773 13
857 14
```

1.4 Searching inside files with `grep`

1. `grep "^>" exercises.fasta`

2. There are 1000 sequences. We can use -c to count the number of matches:

```
grep -c "^>" exercises.fasta
```

Or pipe into wc:

```
grep "^>" exercises.fasta | wc -l
```

3. Yes, three of them:

```
>sequence27 spaces in the name
>sequence52 another with spaces
>sequence412 yet another with spaces
```

One option is two greps piped together:

```
grep "^>" exercises.fasta | grep " "
```

Alternatively, in one regular expression

```
grep "^>.* .*" exercises.fasta
```

4. `grep -v "^>" exercises.fasta`

5. Three. First extract the sequences, then search for n:

```
grep -v "^>" exercises.fasta | grep -c -i n
```

6. Yes, one sequence. Try:

```
grep -v "^>" exercises.fasta | grep -i -v "^[acgtn].*$"
```

Alternatively, we can use the ^ to ask for matches NOT in the alphabet [acgtn]

```
grep -v "^>" exercises.fasta | grep -i "[^ACGTN]"
```

7. 66 sequences. Try:

```
grep -v "^>" exercises.fasta | grep -c "GC[AT]GC"
```

8. We found the total number of sequences earlier:

```
grep -c "^>" exercises.fasta
```

… which outputs 1000

This finds the number of unique names:

```
grep "^>" exercises.fasta | sort | uniq | wc -l
```

… which outputs 999.

Therefore there is 1000 - 999 = 1 name repeated.

1.5 File processing with `awk`

1. Using:

```
awk -F"\t" '{print $1}' exercises.bed | sort -u
```

Should give you:

```
contig-1
contig-3
contig-4
contig-5
scaffold-2
```

2. There are 5 contigs. Use the command from the previous exercise and count the number of lines with `wc`:

```
awk -F"\t" '{print $1}' exercises.bed | sort -u | wc -l
```

3. There are 164 features on the positive strand. Try:

```
awk -F"\t" '$6=="+"' exercises.bed | wc -l
```

4. Ther are 124 features on the negative strand. Try:

```
awk -F"\t" '$6=="-"' exercises.bed | wc -l
```

5. There are 293 genes. Try:

```
awk -F"\t" '$4 ~ /gene/' exercises.bed | wc -l
```

6. 5 genes have no strand assigned to them. Try:

```
awk -F"\t" '$4 ~ /gene/ && $6 != "-" && $6 != "+"' exercises.bed | wc -l
```

7. Yes (6 of them are). First, the number of genes was found earlier:

```
awk -F"\t" '$4 ~ /gene/' exercises.bed | wc -l
```

The number of unique names is:

```
awk -F"\t" '$4 ~ /gene/ {print $4}' exercises.bed | sort -u | wc -l
```

Alternatively, the names can be found like this:

```
awk -F"\t" '$4 ~ /gene/ {print $4}' exercises.bed  | sort | uniq -c | awk '$1>1'
```

9. 18541. Try:

```
awk -F"\t" '$4=="repeat" {score+=$5} END {print score}' exercises.bed
```

10. There are 75 features in contig-1. Try:

```
awk -F"\t" '$1 == "contig-1"' exercises.bed  | wc -l
```

11. There are 12 repeats in contig-1. Try:

```
awk -F"\t" '$1 == "contig-1" && $4 == "repeat"' exercises.bed  | wc -l
```

12. The mean score is 560.833. Try:

```
awk -F"\t" '$1 == "contig-1" && $4 == "repeat" {score+=$5; count++} END{print
score/count}' exercises.bed
```

1.6 BASH scripts

No questions in this section.

1.7 Advanced BASH

1. Here is an example of what this script could look like:

```
#!/usr/bin/env bash
set -e

# check that the correct number of options was given.
# If not, then write a message explaining how to use the
# script, and then exit.
if [ $# -ne 1 ]
then
    echo "usage: example_1.sh filename"
    echo
    echo "Prints the number of lines in the file"
    exit
fi

# Use sensibly named variable
filename=$1

# check if the input file exists
if [ ! -f $filename ]
then
    echo "File '$filename' not found! Cannot continue"
    exit
fi
```

```
# If still here, we can count the number of lines
number_of_lines=$(wc -l $filename | awk '{print $1}')
echo "There are $number_of_lines lines in the file $filename"
```

2. Here is an example of what this script could look like:

```
#!/usr/bin/env bash
set -e
for filename in ../scripts/loop_files/*; do ./exercise_1.sh $filename; done
```

3. Here is an example of what this script could look like:

```
#!/usr/bin/env bash
set -e

# Check if the right number of options given.
# If not, print the usage
if [ $# -ne 1 ]
    then
        echo "usage: example_3.sh in.gff"
        echo
        echo "Gathers some summary information from a gff file"
        exit
fi

# store the filename in a better named variable
infile=$1


# Stop if the input file does not exist
if [ ! -f $infile ]
then
    echo "File '$infile' not found! Cannot continue"
    exit 1
fi

echo "Gathering data for $infile..."


# Gather various stats on the file...


# Total number of lines/records in file
total_records=$(wc -l $infile | awk '{print $1}')
echo "File has $total_records records in total"


# Get the sources from column 2.
echo
echo "The sources in the file are:"
```

```
awk '{print $2}' $infile | sort -u


# Count the sources
echo
echo "Count of sources, sorted by most common"
awk '{print $2}' $infile | sort | uniq -c | sort -n


# Count which features have no score
echo
echo "Count of features that have no score"
awk '$6=="." {print $3}' $infile | sort | uniq -c


# Find how many bad coords there are
echo
bad_coords=$(awk '$5 < $4' $infile | wc -l | awk '{print $1}')
echo "Records with bad coordinates: $bad_coords"




#_________________________________________________________________#
#                                                                 #
#      WARNING: the following examples are more advanced!         #
#_________________________________________________________________#

# if there were records with bad coords, find the sources responsible
if [ $bad_coords != 0 ]
then
    echo
    echo "Sources of bad coordinates:"
    # Instead getting one source per line, pipe into awk again to print them
    # on one line with semicolon and space between the names
    awk '$5 < $4 {print $2}' $infile | sort -u | awk '{sources=sources"; "$1} END{print substr
fi




# Count of the features. Instead of using awk .... |sort | uniq -c
# we will just use awk. Compare this with the above method
# used to count the sources. Although it is a longer command, it is more efficient
echo
echo "Count of each feature:"
awk '{counts[$3]++} END{for (feature in counts){print feature"\t"counts[feature]}}' $infile | s
k2n
```

```
# This example is even more complicated! It uses a loop to
# get the mean score of the genes, broken down by source.
echo
echo "Getting mean scores for each source..."
for source in `awk '{print $2}' $infile | sort | uniq`
do
    awk -v s=$source '$2==s {total+=$6; count++} END{print "Mean score for", s":\t", total/cou
done


# We can use awk to split the input into multiple output files.
# Writing print "line" > filename will append the string "line"
# to a file called filename. If a file called filename
# does not exist already, then it will be created.
#
# Write a new gff for each of the sources in the original input gff file
echo
echo "Writing a file per source of the original gff file $infile to files called split.*"
awk '{filename="split."$2".gff"; print $0 > filename}' $infile
echo " ... done!"
```