1 Linux for Bioinformatics

No exercises in this section.

2 Basic Linux

2.1

2.2

2.3

2.4

2.5

2.6

2.7

2.8

2.9

```
find . -type d
```

2.10 Exercises

Navigate to the basic directory with:

```
cd ~/course_data/linux/data/basic
```

1. `ls -al`

This should show 2 files `directory_structure2.png` `directory_structure.png` and 2 directories `Pfalciparum` `Styphi`.

2. There are 3 files in the directory (`Styphi.fa`, `Styphi.gff` and `Styphi.noseq.gff`). 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 -al Styphi
```

3. There is 1 file in the directory called `Pfalciparum.bed` (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 -al Pfalciparum
```

4. There are 7 GFFs in the linux directory. To search from the linux 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
```

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

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

3 The commands `grep` and `awk`

3.1

3.1.1

3.1.2

3.1.3

3.1.4

3.1.5 Exercises

Navigate to the linux/data/grep directory with:

```
cd ~/course_data/linux/data/grep
```

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

Fasta sequences contain two lines per sequence (a header and then the corresponding sequence). All FASTA headers start with > so we search using ^> which means get me all lines which start with (^) a greater than symbol (>). This will look something like:

```
>sequence1
>Sequence2
>thing3
>Read4
```

2. There are 1000 sequences. We 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 " "
```

Here we look for lines which start with (^) a greater than symbol (>) and then look for any of these lines that contain the space character.

Alternatively, we can do this with one regular expression:

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

Here we look for lines which start with (^) a greater than symbol (>) followed by zero or more (*) of any characters (.) followed by a space followed by zero or more of any characters.

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

We use the -v option which is for inverse matching or returning everything that does not match the pattern we have here in quotes. So in the quotes we are asking for all sequences that begin with '>' but by using -v we are asking for all sequences which do not start with >.

5. Three.

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

Here we use the same command as before to get only the sequences (not the headers). We assume that 'n' is either 'N' or 'n' (i.e. upper or lower case). We could search individually for each of those, but to make it easier we do it in one command with -i which means the pattern search should be case insensitive. We use -c to count the number of matches.

6. Yes, one sequence. Try:

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

Here we use the same command as before to get only the sequences (not the headers).

Alternatively:

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

Here we use the same command as before to get only the sequences (not the headers). Then we use grep with the ^ symbol to ask for matches NOT in the alphabet [acgtn] .

7. 66 sequences. Try:

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

Here we use the same command as before to get only the sequences (not the headers). We use -c to count the number of matches to the strings GCAGC and GCTGC.

8. We found the total number of sequences earlier:

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

... which outputs 1000

To find the number of unique sequence names:

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

... which outputs 999.

Therefore there is 1000 - 999 = 1 name duplicated.

If you want to see which one is duplicated you can use the -c option to tell us the number of times each sequence occurs. Here awk is used to determie which of the counts (first column or $1 is greater than 1 (i.e. is present more than once):

```
grep "^>" exercises.fasta | sort | uniq -c | awk '$1 > 1'
```

This gives:

2 >sequence1

3.2

3.2.1

3.2.2

3.2.3 Exercises

Navigate to the linux/data/awk directory with:

```
cd ~/course_data/linux/data/awk
```

1. Try:

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

Here we use awk to split the file exercises.bed into individual columns and use `-F"\t"` to indicate the columns are to be sperated based on the tab character. We print the first column (denoted by $1) which will be sequence name) and pipe this output to the `sort -u` command which will sort the list of sequence names and -u will select only the unique values from the list.

This should give you:

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

2. There are 5 contigs.

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

Here we use the command from the previous exercise to get the list of unique sequence names and count the number of lines with `wc`.

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

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

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

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

5. There are 33 repeats. Try:

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

4 Advanced Linux

No exercises in this section.

5 BASH scripting

5.1

5.2

5.3

5.4 Exercises

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/coun
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!"
```