README.md

# Class 2 – Sequence data formats

## Goal

- In this module, we will explore different sequence file formats using Unix commands
- We will learn how to perform pattern matching using grep to find files and texts in files
- We will apply for loops to perform same repetitive tasks on different files

## Using GREP for pattern matching

Up until now you've probably accessed sequence data from NCBI by going to the website, laboriously clicking around and finally finding and downloading the data you want.

There are a lot of reasons that is not ideal:

- It's frustrating and slow to deal with the web interface
- It can be hard to keep track of where the data came from and exactly which version of a sequence you downloaded
- Its not conducive to downloading lots of sequence data

To download sequence data in Unix you can use a variety of unix commands such as sftp, wget, curl. In class 6, we will use third party tools that are specifically developed to download data from sequence databases.

Here, we will use curl command to download some genome assemblies from NCBI ftp location:

- Go to the directory on your home computer from which you'd like to work (e.g. home or Desktop)
- Execute the following command to create a new directory in which we will work in, enter it and confirm that it worked

```
#Create the directory
mkdir class2_lab

#Enter the directory
cd class2_lab

#Confirm you are in the directory
pwd
```

- Now get three genome sequences with the following commands:

```
curl ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/241/685/GCF_000241685.1_ASM24168v2/GCF_000241685.1_ASM24168v2_geno

curl ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/409/005/GCF_000409005.1_gkp33v01/GCF_000409005.1_gkp33v01_genomic.

curl ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/165/655/GCF_000165655.1_ASM16565v1/GCF_000165655.1_ASM16565v1_geno
```

***If you are working on gitbash and curl doesn't work, then you can download from the Canvas website***

- To enable quick file transfers these genome assemblies came in compressed format. So, we will use the 'gzip' command with the '-d' flag to decompress the compressed fasta files. One option is to run the command three times (once for each file), but as a shortcut we are going to use '*', which is called the wildcard character. Here we are using '*' to tell Unix to apply our gzip

command to all files ending in '.fna.gz'.

```
gzip –d *.fna.gz
```

The decompressed files are genome assemblies in fasta format. The '.fna' extension typcally indicates that these are nucleotide sequences, as opposed to amino acid sequences, which would be .faa.

***Note that these are really just text files, and the extension is so we can know what's in them and how to work with them.***

Fasta files are a common sequence data format that is composed of alternating sequence headers (sequence names and comments) and their corresponding sequences. Of great importance, the sequence header lines must start with “>”. These genome assemblies have one header line for each contig in the assembly. First, use less to explore the file and remember the trick of using "/" within less to search. In this case, you can search for ">" and use "n" to move to the next sequence header.

Next, let's see if we can learn about the sequences in our file using the Unix command 'wc'. 'wc' stands for "word count", and allows you to count the number of lines, words and characters in a text file. Let's run 'wc' on our assemblies to get a sense of how big our E. coli sequence is:

```
wc E_coli.fna
```

We can see that there are 60091 lines, 60371 words (i.e. character groups separate by space or new line) and 4,866,461 characters. Comfortingly, the E. coli genome is typically around 5 Mb (5 million base pairs), so the number of characters roughly checks out! Although, note that we are including characters in the sequence headers in our count here, so we are overestimating the size of the sequence a bit.

How can we use a single command to run wc on all three of our fasta files?

▸ Solution

Next, we want to figure out a way to use Unix commands to figure out the number of sequences in our file. Unfortunately, each sequence is split up onto more than one line, so we can't just use wc to count the number of lines and divide by 2. So, instead we are going to use the 'grep' command to fish out the sequence header lines and count how many there are. 'grep' stands for 'global regular expression print', and is an extremely powerful tool for searching for patterns in files. Let's now apply grep to search for ">" in our fasta files, as we know that each sequence should start with this character.

```
grep ">" E_coli.fna
```

***Note that we put ">" in quotation marks, as ">" is a special Unix character, which we will learn about imminently :)***

So, now that we pulled out the sequence headers, we just need to count them to determine how many sequences are in our file. We could do this by hand, but we are lazy computer programers and should never do such a tedious task! Fortunately, we just learned a command for counting lines in files - 'wc'. To enable counting the lines, let's put the output of grep into a file by using output redirection.

```
grep ">" E_coli.fna > E_coli_headers

wc E_coli_headers
```

What flag for wc can you use to just print out the number of lines (hint - look in the man page)?

▸ Solution

We now can see that our E. coli fasta file has 35 sequences. Now, a downside of what we just did is that we created this intermediate file containing headers that we don't really need. Ideally, we would be able to apply 'wc' to the output of grep, without going through the intermediate step of redirecting grep output to a file. Fortunately for us, Unix thought of this and gave us the '|' (called pipe). You can think of pipes like a literal pipe, where the output of one command flows into the next one. So, let's string our 'grep' and 'wc' commands together into a single command using a pipe.

```
grep ">" E_coli.fna | wc -l
```

Try this command on other assemblies to see how many contigs they contain.

What happens when you do the following in an attempt to use wildcards to get the number of sequences in each file in a single command? Does it give you what you want?

```
#Number of sequences in each file?
grep ">" *.fna | wc -l
```

## Using for loops to perform same actions on different files

So, using the wildcard in the command above did not have the desired action, as it told us the total number of lines in all files combined, instead of in each individual one. What we need is a way to execute our wc command on each file, but using a single command. You might say to yourself that it isn't a big deal to just run the command three times (once for each file), but what if you had 10 sequences? Or 100 sequences? It becomes inefficient and error prone, so we want to avoid that. What we are going to do instead is use a for loop, to loop over each file and run our command. For loops are not something unique to Unix, and are a fundamental tool for most programming languages for executing repetitive tasks in a streamlined way.

To get a sense for how for loops look in Unix and how they work, try the below example of for loop, that loops over a bunch of numbers and prints out each value until the list is exhausted.

```
for i in 1 2 3 4 5; do echo "Looping ... number $i"; done
```

The above for loop has the following key pieces:

1. for - All for loops start with the word 'for', to indicate the loop command is starting
2. Loop variable - In our case the loop variable is 'i'. We call it i here, but it could be called anything. The loop variable is what is going to change each time through the loop, and allow us to perform a single command in slightly different ways (e.g. run our grep/wc pipe on different files).
3. Loop list - In our case the loop list is '1 2 3 4 5'. The loop list is the set of things you want to apply your command to.
4. do - All for loops have the word 'do', which indicates that you are about to enter the command that will be repeated each time through the loop
5. Loop command(s) - After do comes the command that you want to run. A few things to note: i) this command should always include the loop variable, as that is what is going to change each time the loop statement is executed and ii) when you use the loop variable you need to preface it with a $ (e.g. $i)
6. done - All for loops have the word 'done', which indicates that the loop is ending. The commands between 'do' and 'done' get executed each time through the loop

So, what actually happens when we run the above for loop? Well, the following occurs:

1. The loop variable i is assigned 1 (the first value in our loop list)
2. echo "Looping ... number 1" is executed
3. The loop variable i is assigned 2 (the second value in our loop list)
4. echo "Looping ... number 2" is executed
5. The loop variable i is assigned 3 (the third value in our loop list)

6. echo "Looping ... number 3" is executed
7. The loop variable i is assigned 4 (the fourth value in our loop list)
8. echo "Looping ... number 4" is executed
9. The loop variable i is assigned 5 (the fifth value in our loop list)
10. echo "Looping ... number 5" is executed

OK - now let's create a for loop to count the number of sequences in each of our files!

```
for filename in E_coli.fna Kleb_pneu.fna Acinetobacter_baumannii.fna;
do
grep ">" $filename | wc –l
done
```

It works! One thing that would make it better is if the name of the file was printed out along with the number of sequences. Can you add a line to the for loop to accomplish this ?

Hint - look at how we printed something to the screen in our counting loop.

▸ Solution

## Exploring GFF files

Finally, let's apply grep, wc, pipes and one new Unix command to explore another common sequence file called GFFs. GFFs are used to convey information on annotations of sequences. For instance, they can communicate where in the genome protein coding sequences are found, as well as some details on their function.

The GFF (General Feature Format) format is a tab-seperated file and consists of one line per feature, each containing 9 columns of data.

column 1: seqname - name of the genome or contig or scaffold

column 2: source - name of the program that generated this feature, or the data source (database or project name)

column 3: feature - feature type name, e.g. Gene, exon, CDS, rRNA, tRNA, CRISPR, etc.

column 4: start - Start position of the feature, with sequence numbering starting at 1.

column 5: end - End position of the feature, with sequence numbering starting at 1.

column 6: score - A floating point value.

column 7: strand - defined as + (forward) or - (reverse).

column 8: frame - One of '0', '1' or '2'. '0' indicates that the first base of the feature is the first base of a codon, '1' that the second base is the first base of a codon, and so on..

column 9: attribute - A semicolon-separated list of tag-value pairs, providing additional information about each feature such as gene name, product name etc.

- Use less to explore first few lines of a gff file sample.gff

```
less sample.gff
```

Note: lines starting with pound sign "#" represent comments and are used to document extra information about the features.

You will notice that the GFF format follows version 3 specifications("##gff-version 3"), followed by genome name("#Genome: 1087440.3|Klebsiella pneumoniae subsp. pneumoniae KPNIH1"), date("#Date:02/09/2017") when it was generated, contig name("##sequence-region") and finally tab-seperated lines describing features.

You can press space bar on keyboard to read more lines and "q" key to exit less command.

- Question: Suppose, you want to find out the number of annotated features in a gff file. how will you achieve this using grep and wc? The tricky piece is getting rid of the comment lines, and then counting the rest with wc. Fortunately, grep comes with a flag to return the invert-match, which means you can exclude lines matching a pattern. Look at the grep man page to find the correct flag, and then use it count the number of non-comment lines.

▸ Solution

OK - let's learn one last command that can help us explore tabular text files like gff's. What makes it tabular is that each row has the same number of columns, and each column is separate by the same character, in this case a tab. A natural thing to want to do is to pull out a single column to work with. The Unix command to accomplish this is 'cut'. When using cut you will almost always want to use a couple of flags, including -d (delimiter - what separates each column) and -f (field - which column do you want to pull).

- Question: Let's see if we can increase our pipe complexity by stringing together three commands! Your task is to count the number of rRNA features in a gff(third column) file using cut, then grep and finally wc?

***Note - the default delimiter is tab, so you don't need to specify the -d command***

▸ Solution

- Question: Try counting the number of features on a "+" or "-" strand (column 7).

▸ Solution