Linux Practical 2: File Manipulation

Viewing the contents of a file

Change to the folder where you are keeping your copy of the tutorial files, eg:

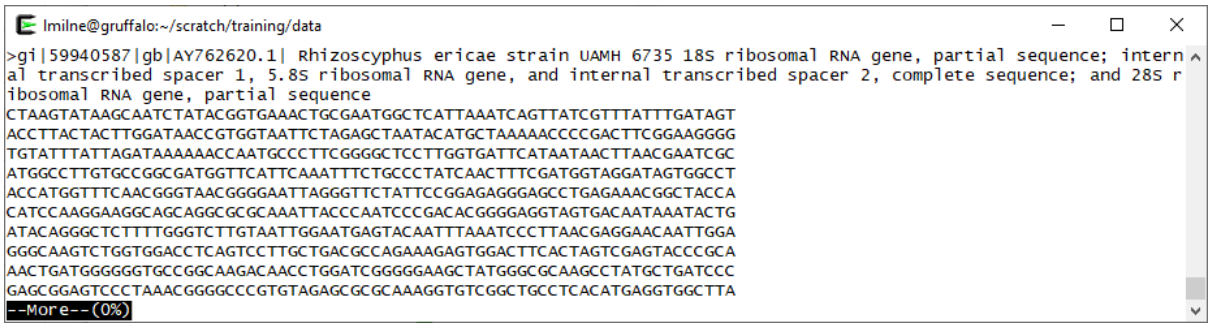
cd /home/*username*/scratch/training/data

Now type the command:

```
cat example_file.fas
```

This is quite a large file, so now try viewing it with:

```
more example_file.fas
```



To use **more**, press the **ENTER** key to see the next page or the space bar to see the next line. Press the **q** key to quit.

When checking a file, sometimes it can be handy to just view the start of the file (perhaps to check that it's really the file you think it is). Try:

```
head example_file.fas
```

You should now see the first 10 lines of the file. Type:

```
head --help
```

…and see if you can use the program's help to generate a command to show only the first 2 lines of the file.

Sometimes you may only want to see the end of a file. Try:

```
tail example_file.fas
```

Again, look at the help for tail work out how to show the last 20 lines of the file.

We can find out how many lines are in the file using the **wc** command. Normally it counts words (wc=word count) but we can get it to count lines with an extra option:

wc -l example_file.fas

Here, the **-l** option ("ell", not "one") tells it to count lines instead of words.

Piping output

You can send the output of one program directly to another by a process called piping.

For example, let's prove that head is producing 10 lines of output.

```
head example_file.fas | wc -l
```

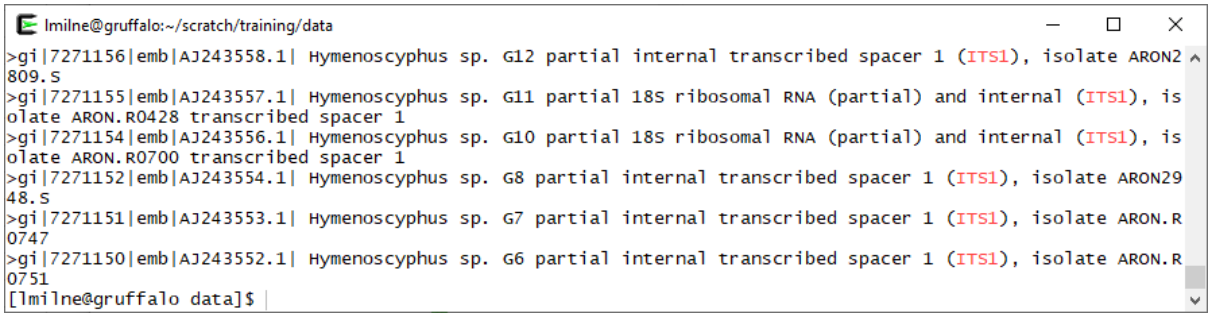
The **|** is the pipe (probably just to the left of your Z key on the keyboard) and it tells Linux to send all the output from the command on its left (**head**) to the command on its right (**wc**). Note that **wc** doesn't need a file name here as it's receiving output from the head command. Hopefully, you'll see the number 10 returned.

Searching for text within a file

To find things in a file you can use the **grep** command. For example, type:

```
grep "ITS1" example_file.fas
```

This should list all lines which have "ITS1" in the line. How many are there? (tip: you need to run it again piping the output to **wc**.)



Hint: Pipe to more to read output easily

If a command produces so much output that you cannot read the first part of it, you can pipe the output to the **more** command to allow you to page through it.

```
grep "ITS1" example_file.fas | more
```

Often you will need the inverted commas around what you are searching for, although in this particular case you could have gotten away with **`grep ITS1 example_file.fas`** – but when searching for things with spaces or special characters the quotes will make it clear what the search term is.

Now search for all lines containing "18S". How many lines are there?

Hint: Careful with grep

The grep command can be used to count the number of lines containing a given bit of text using the -c option.

```
grep -c "18S" example_file.fas
```

Note that using grep to find a line with specific text will return all lines containing the search term, but will not count the number of times the text appears in that line. For example using grep to count the lines containing "stupid" in a file containing a line with "stupid is as stupid does" will only count this line once.

grep searches are usually case sensitive; you can switch this off for a search with the **-i** option:

```
grep -i "hymenoscyphus" example_file.fas
```

Compare the results with the same command without using the **-i** option.

In a FASTA file all the *detail* lines start with a '**>**' – you can search for this as follows:

```
grep "^>" example_file.fas
```

The caret (**^**) symbol is used to mean 'at the start of the line'. The dollar (**$**) symbol can be used to mean 'at the end of the line'.

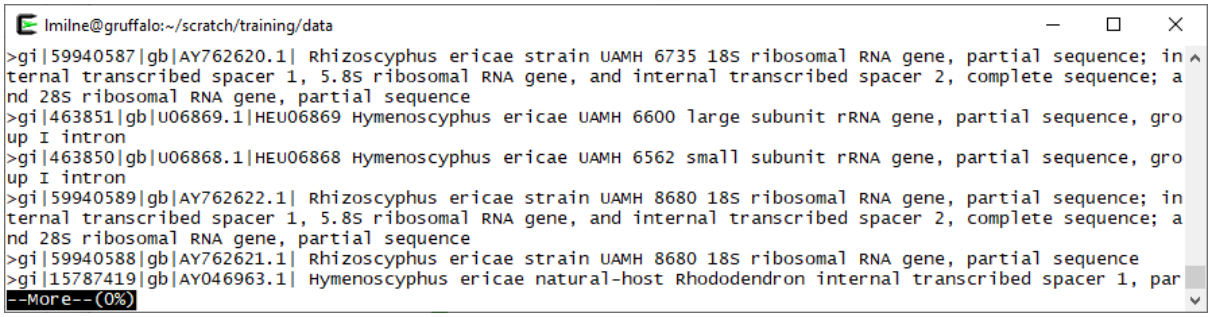
Find out from the **grep** help manual (or **`man grep`**) how to show lines that don't match the search term.

Writing the output to a file

So far we've been letting the commands print all their output to the screen. You can also redirect the output to a file as follows:

```
grep ">" example_file.fas > output.txt
```

Use **more** to look at the contents of the file 'output.txt'.



The special symbol **>** redirects the output to go to a *new* file with the name provided, so be careful because if you give it the name of an existing file it will automatically overwrite it! However, if you *did* have an existing file and you wanted to append (add) the output to that, you could use **>>** instead:

```
grep -i "Hymenoscyphus" example_file.fas >> output.txt
```

Now try creating a file called **descriptions.txt** which has all the *detail* lines from the **example_file.fas** file (be careful with inverted commas here!). How many lines are there in the new file?

Extracting columns of data

The **cut** command is used to extract column data. Let's have a look at the man page for cut. Type:

```
man cut
```

Take a note of the options required to specify the delimiter and the fields.

The data in this FASTA file has been extracted from the NCBI nucleotide database. The description is given as a number of fields of which the 4th is the accession number and the 5th is the description. To extract these columns type:

```
cut -d "|" -f 4,5 descriptions.txt
```

Note that the delimiter character in this example is the same as the pipe symbol which is not ideal, but often you will confront data like this. Once again the quotes help, as they let **cut** know to treat the | symbol as a search term and not a pipe command.

Run it again, this time extract only the accession number and putting the output into a file called accessions.txt

Check that descriptions.txt and accessions.txt have the same number of lines. Have a look at the accessions.txt file, and note that it is not sorted. To remedy this use the sort command as so:

```
sort accessions.txt
```

Look up the man page for **sort**, how would you get the file in reverse order? How would you get it to ignore the case? How could we place the output in a file (other than by redirecting it using ">")?

Let's bring it altogether (hint: with piping). In one command line, extract the *detail* lines from **example_file.fas**, extract the description column, sort it, and then put the result in a file called **descs_sorted.txt**

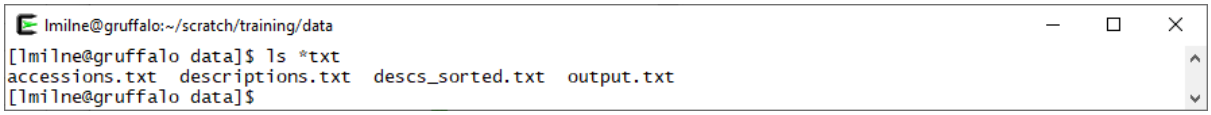
(And if you want a bigger challenge, sort the **descriptions.txt** file by accession number without using cut to extract the accession data.)

Wildcards

Often we want to perform the same function on a number of files at once. First let's see which text files we have, type:

ls *.txt

This will list files that end in '.txt'. The * is called a wildcard and it matches against any character (zero or more times).



Another wildcard is **?** which just matches against any character once. If we had two files file1.txt and file2.csv, **ls file*** would show both files while **ls file?.txt** would only list the first file. Create some empty files with similar names and see how it works. Also try adding the **-l** option to the command as well to see what happens.

Loops

So now we have groups of files, let's do something with them. Try running this:

```
for I in *.txt; do echo "$I has:"; wc -l $I; done
```

Explanation: The semi-colon is a command separator. The first command finds all filenames that match *.txt and places each name into the variable **I** for each iteration of the "for" loop. That starts with the **do**, and ends with the **done**. So inside the loop we have:

```
echo "$I has:"; wc -l $I
```

The echo command displays the text inside the quotes, but what about the **$I**? This is replaced by the current value of the variable **I** , which will be the name of each file as we loop over their names. The final bit should already be familiar; we're going to display the line count of each file.

Now try writing a loop to print the first 5 lines of all the .txt files.