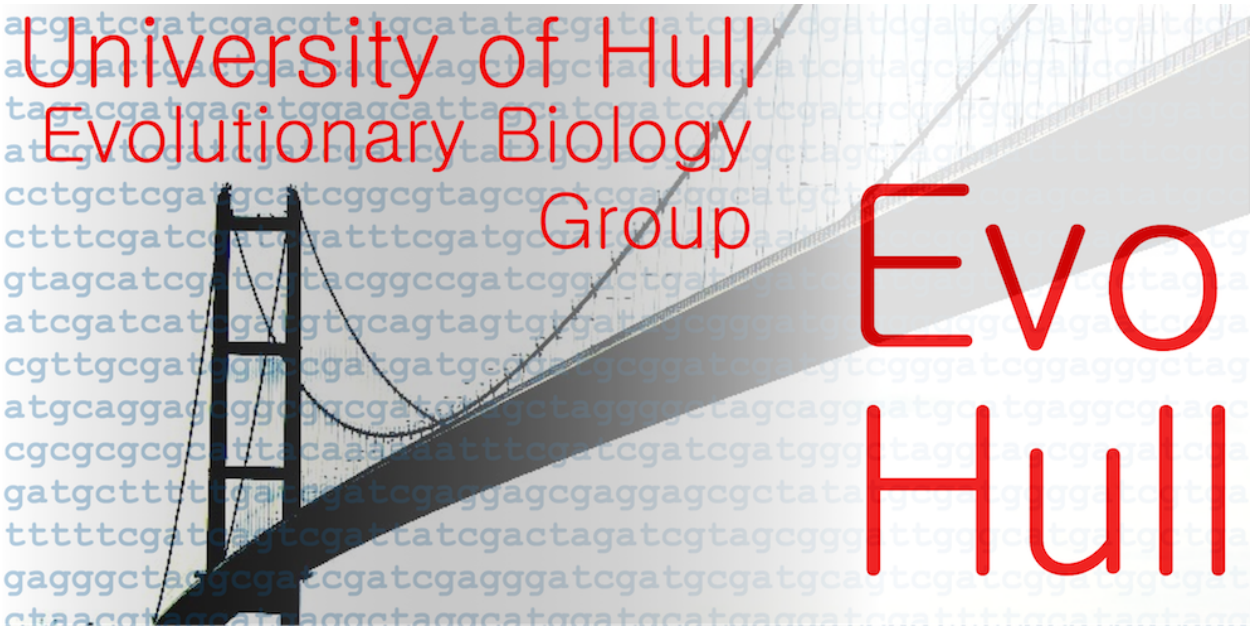
Evolutionary and Environmental Genomics Group @EvoHull

58378, 58412 Environmental Genomics Research Projects



Bioinformatic analysis of invertebrate communities lab manual

Dr Dave Lunt; dave.lunt@gmail.com
Dr Christoph Hahn; c.hahn@hull.ac.uk

Github repository accompanying the course:
https://github.com/HullUni-bioinformatics/egrep-2016

The URL of this document is https://goo.gl/4iMTcj

Table of Contents

1 Aims, background, and approaches

We will begin by discussing the aims of the project. You should know the broad experimental design from this morning's introduction (see the metadata for the samples we will be analysing here). You should think specifically about what you want to know from the DNA analysis, and why, and be prepared to discuss.

1.1 Background thinking/reading

TASK: Research the topic assigned to your pair and present your knowledge to the others in a less than 5 minute informal verbal explanation/chat. Do this research/thinking in the background while you carry out other work. Ask. We will most likely hear the explanations tomorrow in section 4.

1. What is BLAST? How does BLAST actually work?
2. What are MOTUs (molecular operational taxonomic units)? Why are they used and how do we define them in practice?

1.2 Discussion: What kinds of questions can you ask?

We have a dataset with several million DNA sequences to analyse. You should think what it is exactly

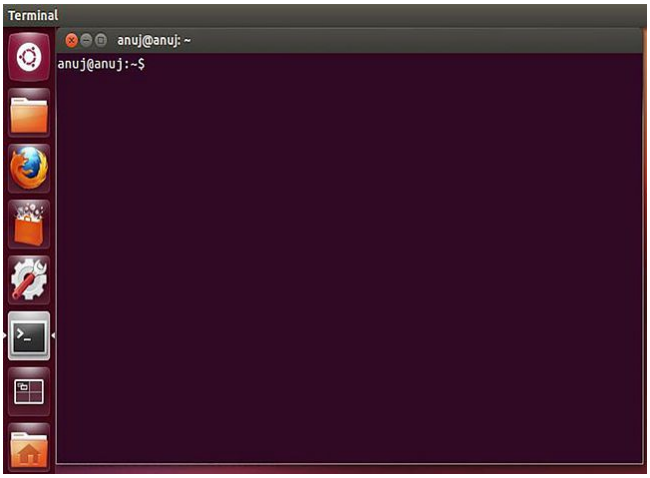
that you want to know. This is really important even for experienced scientists.

- What questions are you going to ask?
- What makes good/bad experimental designs?
- What tests can you perform and how do they
    - broadly investigate the data?
    - answer your specific questions?

2 Introduction to the command line for bioinformatics

There are many reasons to use the command line (also known as the **terminal** or **shell**), a set of skills and an approach to data analysis at the very heart of bioinformatics. In general you can be sure that it is **faster**, more **accurate** (reproducible), and more **powerful**.

In this session you will learn some of the basics of navigating the UNIX-based file system, viewing and searching large data files, and running shell and python scripts. Even these very basic skills will give you completely new ways to work and new analytical options, although obviously this is just a start.



We will be running Ubuntu Linux which is a very popular distribution and we have pre-installed the additional software required for this course.

Below, and in the course, we will give only a very simple overview. **The best way to learn the command line is to Google search for instructions and then try it yourself, experimenting as much as you can.**

Figure: The terminal is an application, you can launch it from the applications menu, dock, or the Ctrl+Alt+T shortcut.

The `$` symbol in the examples below indicates the command prompt, rather than something you type. This may, on some systems be prefixed with additional text, or be formatted differently. You will see something like `metabarcoder@BioPC1:~$`, which refers to the user you are logged in as, before the **@** symbol, the name of the computer you are using and the current directory, after the **:** symbol. The tilde ~ signifies your home directory (more on that in section 2.1 below).

2.1 Navigating the file system

The first thing to do is to learn the basics of moving between places on your computer, checking where you are, checking what files are present and having a quick look at them. Some of the terminology is perhaps slightly new ('directories' rather than 'folders') but using the right words will mean you are speaking the same language as everyone else and make your life easier when Googling for solutions.

Task: as the navigation commands are introduced below try them on your own system

2.1.1 Checking where you are (pwd) and changing directories (cd)

At the command line you need to know "where you are" i.e. which directory (folder) you have open and are working in. Question: If you issue the command to 'list all files' which files will be listed? Answer: Those in the folder where you are currently working, called the working directory. The command to find out where you are is `pwd` short for 'print working directory'. NB: 'print' when working at the command line means 'display on the screen' rather than 'write this to a piece of paper'.

Type the command to display your current directory now (and hit Enter)

You should see something like this, with your command on the line beginning with the `$` prompt and the output on the line below

```
$ pwd
/home/metabarcoder
```

But maybe that isn't where you want to be, in which case you need to 'change directory' and the command for that is `cd`.

```
$ cd egrep-2016
$ cd data
$ pwd
/home/metabarcoder/egrep-2016/data
```

You can see that the `/` symbol denotes levels of directories, so that `'data'` directory is contained within the `'egrep-2016'` directory which is within the `'metabarcoder'` user home directory in the system's `'home'` directory. These *file paths* can be long sometimes but they are always explicit, which is a very good thing for reproducibility. There is no excuse for trying to remember where the data was stored for your analysis, here it is written out, and you will want to record this as part of your experiment.

You can go up one level (to the directory containing your current working directory) by using double dots (ensure there is always a space between the `cd` command and the directory you wish to go to).

```
$ cd ..
$ pwd
/home/metabarcoder/egrep-2016/
```

What happens if you use the `cd` command without telling the system where you would like to change directories to? Try it. How can you find out which directory you are now in?

```
$ cd
$ pwd
```

```
/home/metabarcoder
```

Using `cd` command on its own returns you to your user's home directory, in this case `/home/metabarcoder` from wherever you are.

The tilde symbol (~) is shorthand for this home directory so `/home/metabarcoder/egrep-2016` and `~/egrep-2016` refer to the same directory, which saves a little typing. NB: This is the name of the home directory on Ubuntu Linux. On OS X, for example, it would be something like `/Users/metabarcoder`

Another very useful shortcut is `cd -` (dash) which takes you to the previous directory that you were in. This is really useful when you need to swap between directories that are separated by several levels or that have long names.

```
$ cd ~/egrep-2016/data
$ pwd
/home/metabarcoder/egrep-2016/data
$ cd
$ pwd
/home/metabarcoder
$ cd -
$ pwd
/home/metabarcoder/egrep-2016/data
$ cd -
$ pwd
/home/metabarcoder
```

Make sure that you are actually typing this out for yourself rather than just reading along. This *active learning* will really help it to stick in memory, and come back when you need it, a bit like developing muscle memory.

Above we were issuing commands one at a time; first `cd` then `pwd`. To chain commands together on the same line separate them with a semicolon ie `cd;pwd`.

Try the exercise above again but using semicolons. You should now only need 4 commands not 8.

One of the underlying principles of UNIX is that every item should do one thing (and one thing only) and that we can easily join these items together (usually with ; or a pipe | described later). Think of it like a set of lego blocks that we may put together however we like to build anything we want. Simple units, building complex and impressive outcomes.

We have prepared example data to be used in the following exercises in a directory on your computer - `/home/metabarcoder/egrep-2016/data/exercise-0` .

Navigate to this directory and then confirm that you are in the right place by printing the working

directory path to your screen. Then go back to your home directory before returning to the directory above again to practice your new skills. Test yourself first, but it's OK to review different ways to do this from the manual above. Looking things up is not cheating.

2.1.1.1 Ways to return home

There are 5 ways that you should now know to return to your home directory. Try each to show that they work. The answer is here but discuss first with others, most people only get 2 or 3.

2.1.2 Listing the contents of a directory with ls

A very common thing you will want to do is to display the contents of a directory, i.e. list all the files. You can list the files (and directories) in your working directory using the `ls` command. For this exercise we will be using the qiime tutorial folder. Qiime is pronounced "chime".

Spaces in file and directory names cause difficulties as the shell (called 'bash') treats spaces as the end of a file name. When looking for '`my file`' it complains that it can't find '`my`'.

```
$ cd my directory
-bash: cd: my: No such file or directory
```

Although this can be got around by using quotes `cd 'my file'` replacing with underscores (e.g. `my_file`), hyphens (e.g. `my-file`), or concatenating the words (e.g. `myfile`) are usually better ways to work.. Underscores can be hard to see sometimes in this manual's colour scheme, sorry, but they are always there- never spaces.

```
$ cd qiime tutorial-v1.5.0
$ ls
```

```
./                                  alpha params.tx
../                                 mapping output
18S tutorial files/                 otus
3d biplot/                          qiime tutorial commands parallel.
Fasting Example.fna                 qiime tutorial commands serial.
Fasting Example.qual                split library output
Fasting Example.sff                 wf arare
Fasting Example.sff.txt             wf bdiv even146
Fasting Map.txt                     wf jack
New Map.txt                         wf taxa summary
README
```

Try listing files in long format as `ls -l`

Tip: you can Google to find out what all the data listed means, or use unix's built in manual pages

```
$ man ls
```

Hit the spacebar to advance through the pages. Typing q (quit) will get you out of a `man` page. You can

use `man` with any command, not just `ls`
Googling is not cheating, it's a great way to learn and is highly recommended.

You should go to the directory '**`18S_tutorial_files`**' and investigate what is present- a single fasta sequence file (`.fna`). You can do this of course by changing directory, print your working directory to check where you are, and listing the files present:

```
$ cd 18S_tutorial_files
$ pwd
/home/metabarcoder/egrep-2016/qiime_tutorial-v1.5.0/18S_tutorial_file
s
$ ls
18S_tutorial_sample_seqs.fna
```

How would you do this on a single line command? Show that it works

2.1.3 Some things you will have noticed

Firstly you have to type very carefully, any typo and you will get an error that the file or directory doesn't exist, e.g:

```
$ cd 18S_tutorial_fikes
bash: cd: 18S_tutorial_fikes: No such file or directory
```

A second thing you will have noticed is that some file names are long, complex, and difficult to type without errors.

Tip: you need to learn to use the tab key to autocomplete names.
Real command line gurus use the tab key extensively. If you start typing the command and then hit tab the filename will be auto completed, or, if you haven't typed enough yet to specify a single file (it could be one of several beginning with the same letters) you will probably get a beep, followed by a list of files or directories beginning with those letters. It will also autocomplete the portion of the file or directory name that is shared between them all and wait for you to type more and hit tab again.

Try it now. Navigate to `egrep-2016/data/exercise-4` and list the files present. You should have explored using `tab` to autocomplete the directory names at every level. If not quickly jump back using `cd -` and try again.

2.2 Editing, inspecting, and searching within text files

2.2.1 Editing files

UNIX based systems provide several powerful utilities for editing and inspecting files, either from the command line, or in a simple graphical user interface. You can use the `gedit` program in Linux to open files for viewing and editing in a graphical way, much like **Notepad** or **TextWrangler** in Windows or OSX. Don't start doing this though, it has limitations and is a waste of your time on this course, learn the command line instead. Here we will use the simple text editor `nano` there are many others beyond the scope of this tutorial. The strength of `nano` lies in its simplicity. Help: a beginners guide to `nano`

To view our **`18S_tutorial_sample_seqs.fna`** file in `nano` you should use the general unix approach of `program-name filename`, and assuming we are still in the **`18S_tutorial_files`** directory:

```
$ nano 18S_tutorial_sample_seqs.fna
```

Did you tab complete the name? If not exit using Ctrl+X and try again. Reinforce your skills

TASK: To practice `nano` go to the 10th (Fungi) and 11th (Rhodophyta) sequences and add `/uncultured` to the end of the sequence names. Save the changes using Ctrl-O (called writing-**O**ut), it will ask you if you want to save to the same file, don't, give a new informative name like:
`18S_tutorial_sample_seqs_namefix.fna`

Tip: To close `nano` you should use the Ctrl+X key combination (see ^X Exit in the lower left of the `nano` screen, where the ^ denotes the Ctrl key). You can find nano help pages with a Google search.

To close `less` (below) you should just type q. Using Ctrl+X or q will generally close most UNIX programs, if either of those don't work you can also use the Ctrl+C key combination to kill the program and return to the command prompt.

2.2.2 Inspecting files

`nano` allows us to edit the file in situ, however, if we just want to inspect the file we can display its contents using several other UNIX programs:

1. `cat` to print the whole file to the screen
2. `less` to print the file a page at a time to the screen
3. `head` to print the first few lines of the file on screen
4. `tail` to print the last few lines of the file on screen

One of the problems with DNA sequence files is that they can be large - several hundred megabytes to a few gigabytes is not uncommon. Viewing these files can be difficult, as the files need to be loaded into memory, and can therefore take a great deal of time for the text editor to read from the disk. The `less`, `head` and `tail` commands are very efficient for viewing large files such as these.

TASK: All these commands will be useful for you during this course. You should now try using `less`

`head`, `tail` and `cat` to practice seeing the text file "`C.09.F.fasta`" which can be found in /Data,/exercise-0. Are you using `ls` to see what is available and `tab` to complete the filename? How can you step through the file a screen at a time using `less`? Try Googling for the answer and demonstrate that it works.

2.2.3 Searching within files

Searching within very large files however can be even more troublesome, especially using the standard find functions in a text editor, which aren't optimised for performing searches across very large files. For this reason various tools have been created that allow users to search within large files from the command line, and are highly optimised for their function. One of the most useful utilities for searching within a file is `grep` (**g**lobal **r**egular **e**xpression **p**arser).

`grep` is very simple to use. At the command line you will need to type the word `grep`, followed by the text you are searching for, followed by where (the filename) to look for it. For example, to search for the word **Fungi** in our **`18S_tutorial_sample_seqs.fna`** file we do the following:

```
$ grep "Fungi" 18S_tutorial_sample_seqs.fna
>AY642706_10_1401_Eukarya/Fungi
```

This returns the text `>AY642706_10 1401 Eukarya/Fungi` which is the single line that contains the word **Fungi**.

We can confirm that there is only one by using the *count* flag (`-c`) with `grep` as follows:

```
$ grep -c "Fungi" 18S_tutorial_sample_seqs.fna
1
```

2.2.4 How many sequences do I have?

A very common question is to ask '**how many sequence records are in this enormous fasta file?**' You could of course search for all the greater than > symbols, which is almost certainly the number of records. However you should really search for all the lines **starting with >** rather than the number of times it occurs, as it is possible for a fasta header to contain an internal > . 'Line starts with' is represented by the ^ symbol.

Task: Try to write a grep search to count the number of fasta header lines. Google/ask for help. **Have you remembered the quotation marks around the search phrase?** Unfortunately your solution will probably delete the data file if you forget the quote marks! Why? Discuss

Search the two files `18S_tutorial_sample_seqs.fna` and `C.09.F.fasta` you have already used to determine the number of sequence records. Make sure to discuss your solution.

Tip: you can use the up and down arrows to cycle through your command history. If you find yourself

typing the same command then try pressing the up arrow until you reach the command you want. You can always edit that command if you need to, perhaps using `tab` to autocomplete a new file name. Use the down arrow to bring back more recent commands, and eventually the command line will clear completely, i.e. you are back to the present 'no command'. Type `history` to see a list of all your previous commands or `Ctrl-R` to search them.

2.3 Search, replace, and write output to a new file

`grep` is an excellent tool for undertaking simple yet fast searches within text files. But to search and replace within a text file, or to redirect changes to a new file, we will need to use either `sed` or a simple script (OK there are actually numerous ways of doing this utilizing other tools but this manual will only deal with *simple* examples with `sed` or `python` scripts).

2.3.1 sed the stream editor

`sed` works best when we need to deal with files as single lines, or rows of text data. Since `sed` doesn't try to take the whole file into memory, instead dealing with a line at a time, it has real advantages when files are enormous- as they often are for sequence data.
To search for and replace **Fungi** with **Fungus** in our **`18S_tutorial_samples_seqs.fna`** file we could do the following:

```
$ sed 's/Fungi/Fungus/' < 18S_tutorial_sample_seqs.fna > fungi-to-fungus.f
```

This will replace the single word **Fungi** we identified using `grep` with the word **Fungus**, but output these changes to the file **`fungi-to-fungus.fna`**, leaving the original file unchanged. The `s` within the single quotes signifies this is a substitution command and the `/` characters are delimiters that separate the text to search for, and the text to replace it with. In UNIX based systems the `<` signifies an input, so we are taking input from our **`18S_tutorial_samples_seqs.fna`** file and outputting (`>`) to **`fungi-to-fungus.fna`**.

Tip: Always give meaningful names to files and directories, even if that makes them seem long. The person you are doing this for is 'future you' who will remember less than you think, need clear filenames as one of the ways to make sense of the data, how it has been transformed, and to help record a reproducible experiment. It is very useful to have a filename like:

```
whitby-FDS12763-nematode18S-lenfiltered200bp-uniquespecies.fas
```

Another reason the information-in-filename approach is very useful is that it contains a lot of information you can use for analysis. If you had 1000 files from separate sampling points, you could choose which files to pull data from based on names like "whitby" or "FDS". If you wanted to grab data just from enoplid nematodes from only the Whitby samples you could find and list (con**cat**enate) those with a search, and pipes | to string several jobs together. Below is an example, these files don't exist here, but you are going to try it yourself on files that do.

```
cat ~/allsamples/*whitby*.fas | grep enoplida | sort | uniq -c
```

Task: Google, discuss, and ask until you know what this command does. To help your searches the asterisks are called 'unix wildcards' -why are they used? Think for a moment how much work this single line is actually doing and how long it would take manually?

Above I suggested using the filename
"`whitby-FDS12763-nematode18S-lenfiltered200bp-uniquespecies.fas`"
It may seem an annoying amount of typing to write this much information in filenames, but it isn't *you* who should be doing the writing, it's your script. It's a difficult mindset to overcome, but really useful.

TASK: Go to the `headers` directory. Extract all the header lines from `headers-test.fas` and write to a new file with an informative name. How many are there? Are there any duplicates or are they all unique (`uniq`)? Now repeat this for all the headers containing 2 separate search terms (eg taxon names) that you think of. It doesn't matter what they are. You can examine the headers in the file with your new unix skills to get inspiration. Extra points if you can do this in one single command.
Answer

-PAUSE HERE-

2.3.2 echo

A useful way to write to a text file is with `echo`. This will print to the screen or a file. Here is a short introduction to echo if you need it, although the next sections are fairly self explanatory without it.

Try these commands

```
$ echo Hello world!
```

```
$ echo 'Hello world!' > greeting.txt
```

If the file `greeting.txt` does not exist it will be created. If it does exist it will be overwritten. Check the file now exists (how?), then you can use one of the commands above (`cat` maybe? Do you remember the others?) to inspect the file you have just created. If you wish to append text to a file rather than replace it you can use the `>>` symbol:

```
$ echo 'Hello again world!' >> greeting.txt
```

Try this and check your success. Routing syntax (`>`, `>>`) is general to UNIX and can be used with other programs too. Imagine that you need to add an extra fasta sequence to the end of a big sequence file, the append symbol `>>` will be helpful.

`echo` also allows us to format files correctly if we need newlines or tabs inserted by using the `-e` flag. The tab symbol is `\t` and newline `\n`.

Can you use these to better format `greeting.txt` ?

Try to imagine what the following command will write & discuss with others:

```
$ echo -e "column1\tcolumn2\nRNA\tDNA" > rna-dna-columns.txt
```

Check your success. These commands are useful when writing a lot of data to a file programmatically, and when format is important, which is a very common situation for bioinformatics work. **Remember this example.** You will use `echo` to create one of these files tomorrow.

A common bioinformatics task is to concatenate a lot of individual sequence files into one single file. This is very time consuming to do in a GUI if you have more than a couple of files to open, copy, close, open, paste. The task at the command line however **scales** easily from 1 to 1 million files. You already have all the skills to do this.

Task: go to the `fasta-to-combine` directory. Concatenate all 10 sequence files into a new file with an informative name. Do not add the contents of the README.md file. Demonstrate your success.

Lastly `echo` can write file information like file names

```
$ echo *.fas > fasta-file-names.txt
```

This would write the name of every file in the current directory with a `.fas` extension to a file called `fasta-file-names.txt` which is often very useful when you need to record lots of output file information.

By this stage you have done a lot of work, and learned a lot [Pause]

Take a break and check before proceeding.

2.4 Simple analysis scripts (programs)

2.4.1 Text-processing scripts

Often you will wish to do more complex tasks of manipulating text files. These are best done with simple scripts and most bioinformaticians would use a `python` script to do these sorts of tasks. Learning `python` is not part of this tutorial (even though we run many `python` scripts) but there are many free online courses if you wish to improve your knowledge (e.g. pythonforbiologists.com Google for many, many more).

You have a file **`example-rna.fas`** in the `/data/exercise-0/backtranscribe` directory which holds fasta format RNA sequences. You need to change these sequences to DNA. You could search and replace U with T using `sed` as above (try to write this command for yourself). Unfortunately that will change every U to a T in the sequence headers too. Instead a simple, but much more flexible and intelligent, python script could be used,. This has been written for you called **`RNAtoDNA.py`**

2.4.2 Python scripts

Task: navigate to the correct directory and identify the python script. Instructions are in the Navigating the File System section above if you have forgotten.
Have a look at this `RNA2DNA.py` file using your new command line skills (see Displaying and searching within text files if you have forgotten). **You won't understand everything, that's OK you're not supposed to.** Have a quick guess what some parts might mean and then read on.

Comment lines begin with a hash # symbol. These are just for humans to read, they are ignored when the script is executed (run). Scripts with lots of comments are much easier to understand and you should use them yourself when you write or modify a script.

If you are familiar with fasta format files, and wanted to turn RNA sequences into DNA, you could probably write some pseudocode quite quickly, and one version could look like this:

- `Open the input data file containing RNA sequence`
- `Create an output file with a new name to save DNA sequence to`
- `If input data line starts with > its a header line`
    - `write header line to output file`
    - `move on, we're not changing headers`
- `Otherwise its sequence data`
    - `change all U --> T making it a DNA sequence`
    - `write changed line to output file`
- `Repeat until end of file, close files, claim success`

Now read the python script again, is it more understandable? Some parts may not be obvious, but it's generally like the pseudocode above. Discuss the script with someone.

Task: Create a new DNA fasta file from the file `example-rna.fas` provided in this directory.
In order to run a program or script we specify the program to be run (`python`) and the file to be executed (`RNA2DNA.py`).

```
$ python RNA2DNA.py
```

The new DNA fasta file created has an ugly name, use `ls` to display it. Google how to rename files at the unix command line, and rename it "`new-dna.fas`" or something even more informative. NB remember spaces in filenames cause troubles at the command line, that's why dashes or underscores are commonly used.

You have understood and run a python script, in a unix shell, to reformat nucleotide sequence data. Our work here is done, you are now a bioinformatician, shake hands, welcome to the club!

2.4.3 Shell scripts- collecting together lots of commands

The UNIX based shell allows us to execute **shell scripts**, which have the `.sh` extension. Shell scripts are powerful way to link together lots of different commands and then execute (run) them all at once. Below is a walk-through demonstrating a complex shell script.

In our **qiime_tutorial-v1.5.0** directory we have two shell scripts:

1. **`qiime_tutorial_commands_parallel.sh`**
2. **`qiime_tutorial_commands_serial.sh`**

Its very easy to get lost or panic in the next paragraphs. Don't panic! Just skim this section and ask someone. It's a deliberately complex example, the point of this exercise is not that you read in detail, understand exactly, and remember every detail. Our aim is different, make sure you read the last paragraph in this section.

By doing a `cat qiime_tutorial_commands_serial.sh` we can view the content of this file on screen. The first few lines are:

```
#!/bin/bash
# interactive commands are commented out
#print qiime config.py

# Pre-processing
echo "Check mapping file"
rm -rf mapping_output ; check_id_map.py -m Fasting_Map.txt -o
mapping_output -v
```

The first line (`#!/bin/bash`) is what we call the *shebang* line and points to the location of the shell program we wish to use when executing the program. Any other **lines that begin with a hash (#) are**

comment lines and are ignored by the shell. Comment lines can help in describing what each part does and are therefore very useful to remind yourself, and others, what the script was intending to do.

The first command executed is `echo "Check mapping file"`, which displays the text "`Check mapping file`" on the screen. It then immediately executes `rm -rf mapping_output` to delete any previously created **`mapping_output`** directory. `rm` is a utility for removing directory entries, which includes other directories and files. "Flags" are very useful command modifiers, you used one when you told `grep` to keep a count (`-c`) of the times it found the search term. The `-r` flag here is used to delete all files within a given directory recursively, i.e. including all subfolders and their contents within the specified directory. The `-f` flag removes any prompts to remove files or directories, therefore deleting everything silently, or without displaying any further information on the screen. **Yes this is 'dangerous'**, it's best to use `rm -rf` very cautiously.

The semicolon (`;`) separates commands on a single line. The command `check_id_map.py -m Fasting_Map.txt -o mapping_output -v` is then executed. **`check_id_map.py`** is a python script that checks the mapping file (**`Fasting_Map.txt`**) to ensure it meets the necessary criteria. The `-m` flag signifies the location of the mapping file which, as we are in the appropriate directory already, is simply given as **`Fasting_Map.txt`**. The `-o` flag points to the output directory, given as **`mapping_output`**, for any files that are produced as part of the verification process. The `-v` flag defines *verbose* mode, which means that all of the information that can be printed to screen, will be printed to screen. This is in contrast to *quiet* mode, which stops text being printed to screen.

2.4.3.1 The point of scripts - power, speed, reproducibility

I hope you can see that this shell script has done a lot of complex work in the first few lines already, and the script is much longer. Entering all these commands with the correct flags and file locations from the command line directly would be very difficult, and error prone, especially if you have to do this several times. **Scripts aid reproducibility and save you work.**

Imagine that you were instead in a GUI environment in Windows, and had to click buttons and type into text boxes to change analysis parameters. Just to set the information contained in those few lines of shell script described above would be much more work. What if you had to repeat 3 times, changing one parameter but setting all the others the same? That would be easy with a shell script where you already know how to search and replace text in a text file (like a script) from the command line, but requires a lot of repetition in a GUI. What if you had to do this 1000 times across different combinations of parameters and write informative output filenames to record which parameters were used? Impossible in a GUI but straightforward with only a few basic scripting skills. You could already probably write the pseudocode to do this. You could also give the script to a collaborator to run the same analysis on their data. Or better yet add your analysis script to the massive number already available online for all to use without restriction.

I hope you can see the reasons that bioinformaticians, and most modern biologists working with lots of data, use the command line and scripts rather than proprietary GUI programs.

2.5 Tasks- review of command line skills

Use the skills you have learned to answer the questions below. The information you need is in the sections above but this is 'open book', you can use the internet just like a proper bioinformatician.

1. How many Archaea bacteria are there in the 18S_tutorial_sample_seqs.fna?
2. How many nematodes are there in 18S_tutorial_sample_seqs.fna?
3. What is the last sequence record? (No scrolling down please!)
4. Search for Bacteria and replace it with Eubacteria
5. Demonstrate that you can (a) edit the file directly (b) save the edit as a new file
6. You have been provided with a shell script called `welldone.sh`, in the `scripts` directory, show that you can run it (set your terminal window to a decent size first)

2.6 Some reading if you wish to extend your knowledge

1. UNIX Tutorial for Beginners http://www.ee.surrey.ac.uk/Teaching/Unix/
2. Command line history tricks http://www.thegeekstuff.com/2008/08/15-examples-to-master-linux-command-line-history/
3. Software Carpentry Introduction to the unix shell on YouTube (great short videos)
4. Unix and Perl Primer for Biologists http://korflab.ucdavis.edu/Unix_and_Perl/unix_and_perl_v3.0.pdf
5. Bradnam and Korf. (2012) UNIX and Perl to the Rescue!: A Field Guide for the Life Sciences (and Other Data-rich Pursuits). ISBN-10: 0521169828 ISBN-13: 978-0521169820 http://www.amazon.co.uk/gp/product/0521169828
6. GREP http://www.gnu.org/software/grep/manual/grep.html
7. SED http://www.gnu.org/software/sed/manual/sed.html
8. Software Carpentry Introduction to programming in Python (great short YouTube videos)
9. Python for Biologists http://pythonforbiologists.com
10. Python for non-programmers https://wiki.python.org/moin/BeginnersGuide/NonProgrammers

Task: What is DNA barcoding? Research in pairs what DNA barcoding is, what is the endpoint, how does it work? Approximately what needs to be done in the lab? Approximately what needs to be done at the computer? Take 10 minutes. Make sure you can describe and contribute/lead the discussion.

Intro to ELNs- Dave

3 BLAST-based sequence assignment

One of the simplest and fastest methods to see what else is similar to an unknown sequence from an experiment is to perform a similarity search using BLAST. Exercise 3.1 below is a hands-on introduction to BLAST search and interpretation.

Student presentations from section 1.1

3.1 A basic BLAST search

In the following steps we will perform a similarity search (using the BLAST suite of programs) of (query) sequences against a custom built BLAST database. This may provide information about the taxonomic identity of our query sequences. Can you explain why?

You are provided with cleaned fasta sequences from an experiment sequencing SSU-rRNA from soil mesofauna (see `~/eqrep-2016/data/exercise-1/`) obtained from two different kinds of fields - organic and conventional, designated by the 'O' and 'C' in the filename. These have been filtered for quality and have had the primer and linker sequences removed. Why is this important? The BLAST suite of programs is already installed on your computers.

Open a terminal and navigate to `~/eqrep-2016/data/exercise-1/data/`. Here you will find a file called `97 Silva 111 rep set euk with taxa.fasta`, which contains a set of reference sequences that have been downloaded from the SILVA rRNA database.

Use your Unix skills to have a peek into the file (use e.g. `less`). Before these reference sequences can be searched using BLAST they need to be turned into a BLAST database that can be searched in a structured and efficient manner. The BLAST suite provides a program for that. Run `makeblastdb -h` in your terminal to get some basic information on the options that the program offers, and take a minute to explore them (`makeblastdb -help` will give you more extensive information).
Now, run the following command in your terminal:

```
$ makeblastdb -in data/97 Silva 111 rep set euk with taxa.fasta -dbtype nucl -title custom silva db -out Silva 97
```

Which new files have been created in our working directory? Use `ls -hlrt` to explore the directory

contents - you are already familiar with the `ls` command, but what does the `-hlrt` part do? Find out by typing `man ls`, `ls --help`, or google it. The files constitute our newly generated custom BLAST database.

We can now do a similarity search of our sequence data against this database using the `blastn` program. Run `blastn -h` in your terminal to get an overview on the options that the program has to offer (again `blastn -help` gives you more extensive information).

Set off a BLAST search by running the following in your terminal:

```
$ blastn -db Silva_97 -num_descriptions 50 -query data/C.03.M.fasta
-out C.03.M.fasta.SILVA_blastn.out
```

In the command above we did not specify any particular format so the results will just be written in default format. However, `blastn` offers different formats for outputting the results and we will generate an alternative to the default format. As `blastn` is running in the current terminal open a new terminal and navigate again to `~/egrep-2016/data/exercise-1/`.

Start a second BLAST search to produce results in another output format using the command:

```
$ blastn -db Silva_97 -query data/O.01.M.fasta -outfmt '6 qseqid
stitle pident evalue' -num_alignments 1 -out
O.01.M.fasta.SILVA_outfmt_6.blastn.out
```

Well done! This will take a little while to run (10-20 minutes) - in the meantime you should make sure that you have made a good experimental record in your ELN. Also, you can use your UNIX skills to explore the query sequences. Do this is in yet another terminal window.

Task: Can you determine how many sequences are contained in each of the files? Use your command line skills to find out and record this in your lab book (here if you have forgotten).

3.2 BLAST output

We will now look at the BLAST search that you just ran. You will remember that you started two analyses which provide the results in two different formats. Explore and compare the BLAST output files. What type of data is provided? How can we understand the output? Are any of the command line techniques you learned useful to interpret and interrogate the data?

Task: Determine how many of your query sequences are lucky enough to be most similar to a nematode

i.e. how many query sequences return a best hit annotated as phylum Nematoda? Can you answer this question by using your command line skills on the default BLAST output `C.03.M.fasta.SILVA_blastn.out`?

What about the alternative output format file `O.01.M.fasta.SILVA_outfmt_6.blastn.out`? Does this make your life any easier?

Make some brief notes in your ELN on the appropriate file to use and why.
What can you discover about the taxonomic diversity revealed by the BLAST search? Make notes.
Discussion points in the group:

- Pros and cons of the different output formats.
- Is it enough to know the 'best' hit for a query sequence?
- The importance of 'good' reference databases.

Task; write brief explanations/notes into your lab notebook

1. what is BLAST and how exactly does it work?
2. explain briefly, but with details, how it can be used to identify species. Maybe with a list or sequence, or diagram
3. list all the problems or challenges with this BLAST taxonomic identification approach you can think of

4 Reproducible research & keeping lab records

We are going to have a discussion about reproducible research. Bioinformatic analyses are complex and parameter dependent. Reproducible research is useful, the right thing, and an important part of modern science. Think a bit and get ready to contribute to a discussion of the following

1. What does "reproducible research" mean? What are the consequences to being able to understand exactly what a scientist did from their Methods section, but being unable to reproduce it? What is the difference between replication and reproducibility? Is replicability even worthwhile? I like this video where the first 2 mins is Carole Goble explaining the terminology.
2. How are you going to make your metabarcoding experiment reproducible? Who is the experimental record for, and how will it be used? What should be preserved, how and where?

4.1 Electronic Laboratory Notebooks; ELNs

We will discuss the importance of ELNs, and you will set up your own in preparation for the next investigations you will attempt. At this point we will start using ELNs created for you in Google Docs. If you have a Google account log in and share your gmail address with the instructors. If not it'll be fine.

4.2 Version control with Git

We will be using Git extensively but it's rather behind-the-scenes. This section is background knowledge.

Git is a version control client. Version control is a method of recording all changes to files within a repository (=directory, and nested directories). Having an explicit history of change as your experiment progresses is useful, and makes your experiment transparent and more reproducible. It is useful because if you screw up, or want to return to a previous state to repeat an analysis, git can roll-back to that point in time easily.

Computer scientists and programmers always use version control, scientists are recently appreciating its major benefits. This course repository is hosted on the popular git site GitHub and using git is a very useful thing to teach yourself as a data scientist. I like Try-Git to learn how to use version control. You can browse the github repository for this course.

4.3 Docker containers

Your work is being carried out in a Docker container, again here is some info for background knowledge.

Having the data and analysis scripts openly shared is great. Reproducible

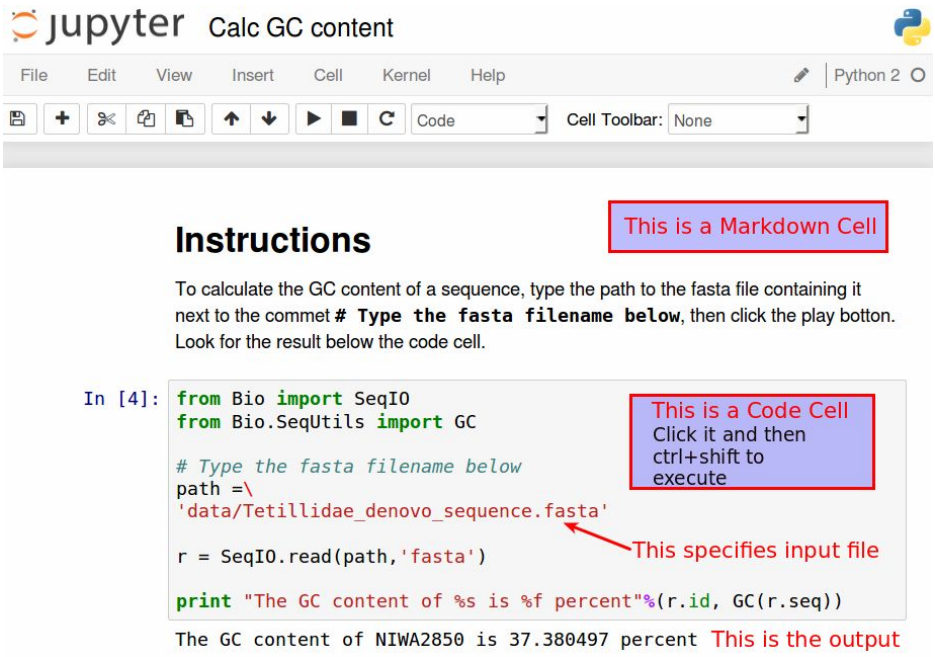
right? Well, hopefully, but what if it doesn't run on your machine because you haven't got a certain program installed? What if when you finally install that program it needs another dependency itself? What if there is no description of the computer system and all its dependencies in the manuscript? One solution to this problem are Virtual Machines, a copy of the entire computer system that can be run within your own operating system. These are very powerful but unfortunately can be very large and unwieldy.

Docker is a leading solution. In short, rather than replicating the entire operating system it tries to be more lightweight and only provides the 'other stuff you need' giving you the same operating system without duplicating the common parts.

In practice Docker makes life much easier, and we have deployed identical instances of the set up for this course on all the machines by using Docker containers. You will work inside a container. So we have (kind of) tested all the analyses in this course, and you are going to replicate/reproduce our experiments as you learn. Docker is therefore a very powerful component of reproducible science since you will start with a system identical to that on which they were first run. If you modified the environment by installing new analysis software the modified Docker container could be archived as part of the publication of your experiment.

4.4 The Jupyter notebook

All scientists should keep a lab notebook of their experiment. When working at a computer however a paper notebook is just silly, you want to be able to copy and paste what you did into an electronic



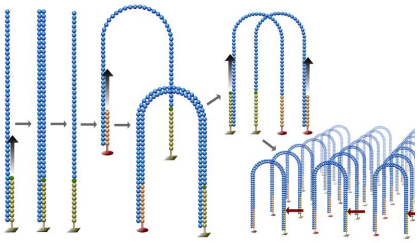
document and insert a picture of the results surely. Well yes, but that is very old-style, and hard work. The Jupyter notebook (the notebook previously known as IPython) is a mixture of a notebook where you record stuff, type your thoughts, and an environment (like the command line) where your actual

work is done. The advantage is that everything is in one place, the commands you ran, the code, the error messages. No copying and pasting results and pictures. Very many bioinformaticians are now using this environment to aid reproducibility, and it is a friendly and time saving solution.

Our analyses in this course are often run from a Jupyter notebook.
When carrying out the analyses you need to fill out an experimental record. A good way to do this, different from the ELN approach, is to use a Jupyter notebook. You can fill in notes, give it a title, perhaps an aim, briefly write a sentence or two about an analysis, then **paste in the code** to a code cell and run it. Maybe then annotate the output with comments on what went right or wrong. This mixture of comment and code makes for a powerful reproducible record when you are working with scripts. Don't forget that the person most likely to want to reproduce your work is future you, a good clear record will pay back.

5 NGS data

In recent years a number of new approaches for nucleotide sequencing have been developed and this type of *massively parallel* generation of sequence data is commonly referred to as *next-generation sequencing* or NGS. It makes little sense to name it this way, but the terminology has stuck.



Different sequencing instruments have specific error profiles, which are important to understand before getting started with analyses. The Illumina sequencing platform is currently the most widely used and a great option for metabarcoding projects with large amounts of data produced at relatively low cost. This part of the course is therefore focusing on Illumina data and is meant as an introduction to basic sequence quality assessment, -filtering and general processing of raw NGS data for analyses in metabarcoding studies.

Figure 2.1: Illumina data generation. See Illumina sequencing videos on YouTube.

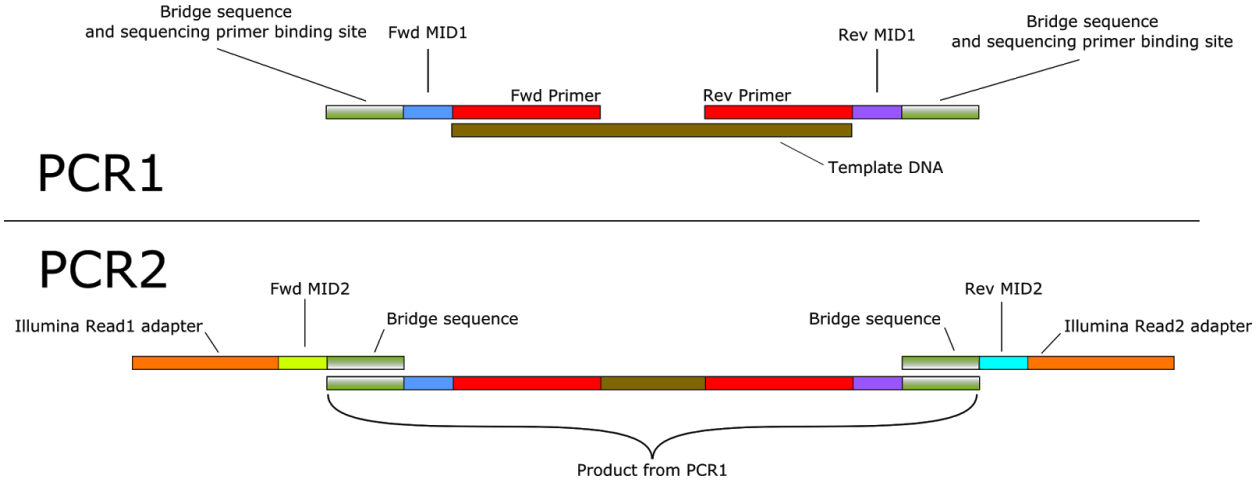


Fig 2.2: Construction of Illumina sequencing library using PCR, illustrating the additional sequences added to the template DNA.

5.1 Starting a Jupyter notebook

Many of the following exercises will be run in a **Jupyter notebook** and this section will get you started. We will be using a number of different pieces of software, so for the sake of reproducibility and to make your (and our) life easier we have set up a self contained environment (called a container) in a docker image (see here for the image repository and here for the underlying GitHub repository), that provides all the necessary software for our analyses. This image is building on the ReproPhylo environment (see also here) that has been developed in-house. Docker is installed on your computers and we will now enter the docker container by running a single script in your command line. The script is present in the `scripts/` directory:

```
$ cd /home/metabarcoder/eqrep-2016/scripts/
```

Navigate there and execute the following:

```
$ ./start metaBEAT nb /home/metabarcoder/eqrep-2016/ --xt
```

This should open a new tab in your browser window and the first thing you will see is a privacy error, which you will need to sort out by clicking on 'Advanced', followed by 'Proceed to localhost (unsafe)'. Now you will need to provide a password, which is 'password'. This should finally open a Jupyter notebook in your browser and place the root of the notebook in your `/home/metabarcoder/eqrep-2016/`. The Jupyter notebook will display the file structure in your `/home/metabarcoder/eqrep-2016/`. Now you are good to go and may forget about the terminal for a while. Open a new notebook and start exploring.

NOTE: You can perform the same terminal operations that you did before in the code cells in the notebook. The cells will however natively interpret commands to be in the python programming language so to use the shell commands that you are already familiar with from your previous exercises you will need to add an exclamation mark `!` before a command to tell the cell that it is a shell command. Try out some of the commands you have used before (e.g. `ls`).

Imagine that this is your lab notebook. In addition to command or code cells you can also write normal text, or formatted markdown text, into cells. You can put subheadings and even embed images. This is a good way to write explanations for Future You of what you did.

5.2 Sequence Quality

One of the primary tasks before analysing your data is to filter and trim it appropriately. This will remove the primer sequences introduced during the experiment (see Fig. 2.2) and filter out bad

sequences. You need to load the data, examine the properties of the data, make lab book entries recording this information, and include or exclude sequences based on criteria of "good/bad" that you understand.

Question: What does bad mean? What might make a sequence good or bad?

Illumina sequence data comes in the so-called 'fastq' format, which can be seen as an extension of the 'fasta' format that you are already familiar with from your tasks yesterday. In addition to nucleotide sequences fastq files also contain quality scores associated with each nucleotide. These describe the probability that a given nucleotide has been called incorrectly and are expressed as phred scores. A phred score of 30 (often used as 'gold-standard') indicates a probability of 1 in 1000 for a base to be miscalled (ie 99.9% accuracy). Phred scores are coded as single characters in ASCII format.

5.2.1 Fastq sequence format

Let's take a few minutes to get familiar with the fastq format by exploring data files that we have prepared for you. We will be doing this in a Jupyter notebook. If it's not already running, please start your Jupyter notebook servers now (see here in case you have forgotten). Navigate to `~/egrep-2016/data/exercise-2/` and start a new notebook. We have prepared a notebook for you here, that should serve as inspiration for solving the below tasks, in case you need it.

Fastq files usually come in compressed form (usually 'gzipped' indicated by a `.gz` suffix). Many programs are able to process gzipped files automatically but others are not, so make sure you know how to de-compress such files (`gunzip`). Alternatively there are ways to display the content of gzipped files without actually writing a much larger de-compressed version of your data onto your disk.

Task: Find two ways to look at the first 12 lines of the compressed file (gzipped) that we provide in `example.fastq.gz` and record them in your notebooks for future reference.

Now that you had your first look at a fastq file can you find at least two differences between fasta (the file format you have been exploring yesterday) and fastq files in terms of formatting conventions? You remember that in fasta format sequence headers (the first line of every sequence record) were characterized by ">". What about fastq? How many lines does a single sequence in a fastq file typically have?

Task: Given the general characteristics of fastq files can you come up with at least two ways to determine the number of sequences in the file `example.fastq` using your command line skills?

The typical length of Illumina sequences ('reads') is currently 100-300 bp and data usually comes in paired-ends, i.e. the same DNA fragment is sequenced from both ends. The 'forward' and 'reverse' reads are mostly provided in two separate files which usually contain '_1' (forward reads) or '_2' (reverse reads) in their filenames. The first forward read pairing with the first reverse read and so on. The pairing information will also be encoded in the read headers. Depending on the length of the

actual fragment the read **pairs might overlap**.

Task: Display the first 8 lines of the files `AHA_ASH_1.fastq.gz` and `AHA_ASH_2.fastq.gz` and explore in order to understand how read pairing works in Illumina paired-end data.

Question: What might an 'interleaved' fastq file be?

Next we'll move on to quality assessment and quality trimming/filtering of Illumina data.

Bonus Task: If you have time, find out what the difference is between 'phred-33' and 'phred-64' fastq sequence quality encoding. Which of the two is used in your fastq files?

5.2.2 Quality and length filtering

For the initial quality assessment of your Illumina data we will use a tool called FastQC. It's almost a classic and is installed locally on your computer. You can start the program by simply running the following in a terminal:

```
$ fastqc
```

The program opens and you can directly load the paired end Illumina data that we've provided in `~/egrep-2016/data/exercise-2/` (`AHA_ASH_1.fastq.gz` and `AHA_ASH_2.fastq.gz`). After a few moments FastQC will provide a report characterizing your data in several ways. Explore the report if necessary with the help provided here. Your data has been produced on a MiSeq sequencing instrument - Illumina's 'Desktop sequencer'. What do you think about your data in terms of 'per base sequence quality'? - Compare to example reports here - would you say the data quality of your files is acceptable? Are there differences between your two files? Discuss and write down conclusions in your lab notebook.

Now, we will perform basic quality filtering/trimming using a set of simple but powerful programs provided in the FASTX-toolkit. See here for some more details on the available programs. They are all installed in your docker container and as usually you can get information on the options by using the `-h` flag with the command. Try e.g.:

```
$ fastq_quality_trimmer -h
```

Task: Quality trimming - Find the appropriate FASTX tool (see list of available tools here) to perform basic quality trimming. End-trim your forward and reverse reads to a quality of Q > 30 and discard all reads shorter than 250 bp. How many reads were retained?

NOTE - The FASTX tools per default interpret phred scores as phred-64. To specify otherwise the programs take a `-Q 33` flag, but this is not reported in the help (`-h`).

Task: Quality filtering - Find the appropriate module and discard any read that has a quality score of Q < 30 across more than 25% of its length.

Explore the effects the quality trimming had on your data using FastQC and make notes in your lab notebook.

Advanced Task: Combine the two commands above using the "pipe" symbol '|' to perform both tasks in a single operation without writing intermediate data to a file.

Possible solutions to the above tasks can be found below or in this Jupyter notebook:

Quality trimming:

```
fastq quality trimmer -Q 33 -t 30 -l 250 -v -i AHA ASH 1.fastq -o AHA ASH 1-trimmed.fastq
```

Quality filtering:

```
fastq quality filter -i AHA ASH 1.fastq -Q 33 -q 30 -p 60 -v -o AHA ASH 1-filtered.fastq
```

Combined:

```
fastq quality trimmer -Q 33 -t 30 -l 250 -v -i AHA ASH 1.fastq | fastq quality filter -Q 33 -q 30 -p 60 -v -o AHA ASH 1-trim-filter.fastq
```

5.3 Sequence redundancy

Another highly recommended thing to do, and especially useful for metabarcoding projects, is to reduce redundancy in your data by read clustering. CD-hit and vsearch are popular approaches. The analysis pipeline that you will use later this week will incorporate sequence clustering to reduce redundancy using vsearch.

Can you explain what redundancy is and why it might be a good idea to reduce it (think of MOTU's)?

We have provided a file containing quality trimmed, cleaned and merged paired-end sequences for you - see `AHA ASH.extendedFrags.fastq.gz`. Let's cluster the sequences at 97% similarity.

First de-compress:

```
$ gunzip AHA ASH.extendedFrags.fastq.gz
```

Then convert from fastq to fasta format:

```
fastq to fasta -i AHA ASH.extendedFrags.fastq -Q 33 >
```

```
AHA ASH.extendedFrags.fasta
```

Finally cluster the sequences using vsearch (run `vsearch -h` to get information on the available options):

```
vsearch --cluster_fast AHA ASH.extendedFrags.fasta --id 0.97
--centroids AHA ASH.extended.clustered-0.97.fasta
```

What did we gain from clustering? How many sequences did you have initially? How many clusters were retained?

Task: Now cluster your input sequences at 95 % similarity.

Discussion:

- Compare 99% vs. 95% clustering.
- Which similarity setting do you think would be appropriate - remember our discussion on MOTUs?

Task. BLAST the non-redundant set of sequences that you got from vsearch (clustering threshold 97%) against the complete nucleotide database from Genbank. Report a maximum of 50 hits per query, otherwise use default settings. The database is prepared for you at `../reference-dbs/BLAST/nt/nt`.

This BLAST search is going to run overnight - We will examine the results tomorrow - **Well done!**

If you haven't done so yet, now would be a great time to update your lab notebook!

5.4 Homework:

5.4.1 MEGAN

There will be a brief discussion of the MEGAN program. You should read the paper and be able to discuss what the program does. You can read the paper in groups, discuss it, Google for other people's opinion, whatever you like. Your discussion should not be of settings and run parameters, but instead of how it analyses the data and what analyses it provides to us, e.g. LCA. MEGAN references are Huson et al 2007, 2011 and are on Canvas in the Files folder. Why not develop a reading team to minimise your work and maximise the coverage of the literature. Make sure that everyone can feed in productively to an overall discussion.

5.4.2 Diversity measures

Later on you will be asked to prepare an understanding, and give a description of diversity measures including: Alpha diversity, Beta diversity, gamma diversity. Also analysis techniques including Rarefaction curves and Principal Component Analyses (PCA). You may wish to start reading now.

5.4.3 Questions to ask

You will soon be asked to come up with a list of questions you could address using the data and approaches you have been introduced to in this course. Thinking about this now when you have time would be a good idea.

6 Tree invertebrate metabarcoding data

6.1 Illumina read processing

In the following sections we will use metaBEAT, a tool tailored towards reproducible and efficient analyses of metabarcoding data that we have developed in-house. It is still under active development and will likely be extended further in the future. The pipeline is available in a Docker container with all necessary dependencies. The Docker image is building on ReproPhylo (Szitenberg et al. 2015). The metaBEAT tool is designed as a wrapper around a complete analysis from raw data, performing (optionally) de-multiplexing, quality filtering, clustering along the way, to denovo OTU tables in biom format. It also supports downstream taxonomic assignment (next section).

In the next exercise we will use the metaBEAT pipeline to process the data from 39 invertebrate metabarcoding samples collected from three UK forests (see sample metadata here) - from raw Illumina reads, via quality trimming, read merging, and clustering, to a de novo OTU table.

Again we provide a notebook that will guide you through the process. Navigate to `~/egrep-2016/data/exercise-3` , start a new notebook and follow the steps outlined there.

6.2 MEGAN

MEGAN student discussion

MEGAN (Huson et al. 2011) is an easy to use tool to extend the interpretation of the BLAST output and represent the taxonomic content of the whole BLAST output.

MEGAN is installed on your computers. Find and start the program by clicking the icon in the panel on the left hand side of your screen or using the search button present in the top left corner of your Desktop panel. Load the result of yesterday's BLAST search (should be in `~/egrep-2016/data/exercise-2/`) into MEGAN using "`File -> Import from BLAST`" from the MEGAN menu. Loading the file into MEGAN will take a few moments.

Based on the results of the BLAST similarity search MEGAN assigns the unknown sequences to nodes in a taxonomic tree following the NCBI taxonomy. The size of the tree nodes corresponds to the number of sequences assigned to the respective node. Information on nodes can be obtained by left-clicking on them. MEGAN also allows you to inspect the individual sequences assigned to a given node (right-click on a node) and furthermore collapse/uncollapse nodes to display the taxonomic

assignments on lower taxonomic levels (right-click on the rightmost node and select `uncollapse subtree`). If no unambiguous taxonomic assignment can be obtained due to conflicting BLAST hits MEGAN will attempt to assign a conservative lowest common ancestor (LCA) to sequences.

Task: Right click on an internal node and `inspect` the BLAST results for a given sequence. Take a second to investigate the assignment MEGAN has made based on what you already know about LCA.

Explore the options that MEGAN is offering for summarizing the taxonomic composition of this sample based of your BLAST results. Try to calculate diversity indices at different taxonomic levels (e.g. `Options -> Shannon-Weaver `).

We have run BLAST for further samples and provide results for some in the directory `~/egrep-2016/exercise-4/`. These files have been converted to a native MEGAN format (*.rma) so they can be loaded directly by simply using `File -> Open`. Find a text file containing metadata for every sample in `~/egrep-2016/data/metadata/2014_tree_metadata.csv`. You can view the file in a text editor or import it into LibreOffice (Ubuntu's free alternative to Excel - green icon in your Desktop panel), or have a look here.

MEGAN also allows the comparison of different datasets - simply load several BLAST results into MEGAN and then select `Options -> Compare `.

Task: Think about the questions you want to answer, discuss, and combine samples into comparisons accordingly.

Explore your options and extend the group discussion on the questions you might ask/answer using MEGAN. Don't forget to record any conclusions in your lab book. Extract figures and/or screen-shots now for your presentations.

7 Comparative taxonomic assignment

In this section of the course we will attempt taxonomic classification of the de novo OTU sequences.

Yesterday you ran the metaBEAT pipeline. Describe clearly what tasks you set it to carry out, and why, to your partner? How has your data changed from the sequencer to this point? Make this clear verbally and in your ELN.

We will again make use of the metaBEAT pipeline. It will evaluate our data using two different approaches to taxonomic assignment:

- BLAST-based LCA
- K-mer based classification using Kraken

You are already familiar with the concepts of BLAST-based LCA from earlier in the course. Kraken was originally developed for 'ultrafast metagenomic sequence classification using exact alignments` (Wood and Salzberg 2014). The approach breaks both the reference data as well as the queries into short sub-sequences of length k (aka k-mers). In the Kraken database all k-mers are labelled with taxonomic information (LCA) based on the species the particular k-mer was detected in. Query sequences are classified based on perfect k-mer matches with this database. Limiting the search only to perfect matches between short sequences is highly efficient, which makes Kraken extremely fast, compared to other approaches. K-mer based strategies are also used by many de-novo genome assembly algorithms.

Your Jupyter notebook servers should still be running. Otherwise start them up as described here. Navigate to `~/eqrep-2016/data/exercise-5`, start a new notebook and follow the instructions here.

Once set running, metaBEAT will generate taxonomically annotated OTU tables in BIOM format from the two different assignment approaches in about 30 minutes. BIOM format has been developed as a standardized format for representing 'biological sample by observation contingency tables'. It allows for rich annotation with metadata and can be interpreted by a number of software packages, such as Qiime, mothur, and MG-RAST. These packages offer extensive functionality for quantitative community analysis and hypothesis testing that exceed the scope of this basic course for now.

Once BIOM format files have been produced we are going to load them into the PHINCH website to examine the diversity of communities. in case you don't want to wait for your own run to finish you can use the *.biom files in the directory `~/eqrep-2016/data/exercise-5/results_backup`). Spend some time investigating and making notes of the options available. Phinch visualizations are quite fancy so it may be worth taking a few snapshots for your presentations. What are going to be the most useful comparisons?

8 Diversity analyses

In order to measure and compare the level and type of diversity within and between locations we need measures of that diversity.

Students to research the following topics in pairs:

- Alpha diversity and Rarefaction curves
- Beta diversity and gamma diversity

Bonus topic

- Principal Components Analysis (PCA)

You should prepare to discuss what these topics are, and how they are used in community metabarcoding.

Add the best resources you find here to this list: yes edit the manual!

- https://www.quora.com/What-is-the-difference-between-alpha-beta-and-gamma-diversity
- https://methodsblog.wordpress.com/2015/05/27/beta_diversity/
- http://setosa.io/ev/principal-component-analysis/
- https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/

Statistical tests of differences between treatments is an important topic. You will need to understand and be able to analyse your data statistically before your final manuscript is submitted. Otherwise you will just be showing a picture and saying that it looks a bit different, or not. Try to narrow down your thinking to very specific questions and then think about how you could test them.

We have performed a number of basic diversity analyses, investigating e.g. rarefaction, richness, and community composition (NMDS) that you can use to aid your discussion and for future reference, here. In our analyses we have used the R packages:

- picante
- rich
- vegan

8.1 What questions should we ask?

Students to list questions and to plan methods to resolve them. Student-lead group discussion culminating in making a list of questions to answer.

8.2 VEGAN

We will be using the R-package VEGAN to carry out many of analyses.

8.3 Detecting Invasive Species

The following species are not native to the UK and are considered a threat, sometimes a very significant threat, to our natives. Familiarise yourselves with them and their status in the UK and surrounding areas.

1. Asian hornet, *Vespa velutina* (just discovered in UK this week!)
2. Citrus longhorn beetle, *Anoplophora chinensis*
3. Harlequin Ladybird, *Harmonia axyridis*
4. *HCLM Cameraria ohridella*
5. *Aedes albopictus*
6. Asian longhorned beetle, *Anoplophora glabripennis*
7. *Aphis gossypii*
8. Cotton whitefly, *Bemis tabaci*
9. Western corn root worm *Diabrotica virgifera virgifera*
10. Mediterranean fruitfly *Ceratitis capitata*
11. Colorado beetle *Leptinotarsa decemlineata*
12. Serpentine leaf miner *Liriomyza huidobrensis*
13. African cotton leaf worm *Spodoptera littoralis*

Task: Determine if any of these are present in any of our samples. Discuss experimental design and confidence in your answer. What are the pros and cons of environmental monitoring using metabarcoding?

9 Results, figures, and data wrap up

Friday am

- examine output from metaBEAT run and load biom data into PHINCH
- questions and answers- what do we now know?
- development of key figures for reports and presentations

Friday pm

- students to practice explanations of our results & conclusions
- more work on tables and figures
- feedback on lab books before assessment PHINCH

Today is the final day of your project, and it is one of the most important days. You will work with the other students to make sure that you understand the experiment, how you have analysed the data to achieve your results and what it all means. Work on getting excellent figures and tables. Archive

figures and tables on eBridge for all students to access. Today you should focus on preparing for your assessments. Firstly is your ELN in good shape, have you got feedback on it? Secondly can you explain your experiment and results in the Science Symposium later in the semester. Being able to explain clearly and briefly is not the same as understanding, they are different skills.

We will work on questions and answers. What questions have we asked? What answers have we been able to provide. We have been writing these on the whiteboard, but students should think of more, and be prepared to discuss and come up with answers, or at least how we might answer.

9.1 Laboratory notebook

- Are all sections completed?
- Do you have a good sequence of intent, action, outcome for each analysis?
- Do you make good use of subheadings, tables, figures, lists?
- Will it be clear in 6 months time?
- Where are the data stored?

9.2 Poster and talk in week 10

- Draw out plan on plain paper/powerpoint
- Book Group-B student meeting into diary as prep for that presentation

9.3 Conclusions and their presentation

Students to explain/present what the results mean to each other. All to contribute

- 30 second summary of results
- pick 1 small result and rehearse its explanation/conclusions
- Students to identify, list and explain key questions
- Students to update what we can now say about them
- Identify concerns, qualifications, work needed
- Identify possible misconceptions of what data means

9.4 Experimental critique

In what ways would you change the experimental design? What aspects of our data analysis are open to question? Have we got a rough assessment of species quantity or an accurate one? What about taxonomic assignment? What about biological and technical replication, is it enough?

9.5 DNA metabarcoding

Discussion of the ways in which metabarcoding brings advantages that conventional taxonomy does not. What are the real differences between the approaches in how they generate data? Scope of taxa

sampled? Why might we expect your data to be different/similar to that of Group-A?

10 Finish

11 Answers

11.1 Five ways to return to your user home directory

1. $ cd home/metabarcoder
2. $ cd
3. $ cd ~
4. $ cd ..;cd ..
5. $ cd -

NB the first 3 are pretty generic and will work wherever you are in your file system. Number four just moves up 2 levels, so will only work if you are 2 levels down from your home directory. Number 5 returns to the previous directory you were in, which in this case was your home. Return to question

11.2 sed the stream editor unique headers

I think this will do it, if you were searching for the unique headers containing "enterolobii" which is a species of nematode

```
cat headers-test.fas | grep enterolobii | sort | uniq >ento-uniq.txt
; wc -l ento-uniq.txt
```

In reality you might take a different approach and split the 'count unique enterolobii headers' and 'write to a file' into separate jobs. Return to question

Caporaso, J.G. et al., 2010. QIIME allows analysis of high-throughput community sequencing data. *Nature methods*, 7(5), pp.335–336. Available at: http://dx.doi.org/10.1038/nmeth.f.303.

Huson, D.H. et al., 2011. Integrative analysis of environmental sequences using MEGAN4. *Genome research*, 21(9), pp.1552–1560. Available at: http://dx.doi.org/10.1101/gr.120618.111.

Koski, L.B. & Golding, B.G., 2014. The Closest BLAST Hit Is Often Not the Nearest Neighbor. *Journal of molecular evolution*, 52(6), pp.540–542. Available at: http://link.springer.com/article/10.1007/s002390010184.

Porter, T.M. et al., 2014. Rapid and accurate taxonomic classification of insect (class Insecta) cytochrome c oxidase subunit 1 (COI) DNA barcode sequences using a naïve Bayesian classifier. *Molecular ecology resources*, 14(5), pp.929–942. Available at: http://dx.doi.org/10.1111/1755-0998.12240.

Wang, Q. et al., 2007. Naïve Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. *Applied and environmental microbiology*, 73(16), pp.5261–5267. Available at: http://dx.doi.org/10.1128/AEM.00062-07.