



Nov. 2-5 2017 Graz

Course material for the practical afternoon session

by

Fernando Fernández Mendoza, Christoph Hahn & Philipp Resl

1. Preparations

Before the workshop:

In order that the practical session run without problems it is important that everyone has the same software installed and all the example datasets ready. Therefore we will be using BioLinux, an Ubuntu based Linux variant that comes with many popular programs used in bioinformatics analyses. Because not everyone will have Linux experience we will be running Biolinux using VirtualBox. VirtualBox is an a free hypervisor software which enables you to run Linux from your own operating system (Windows, Linxu, Mac) like it was a program on your computer. The program you will run will be provided by us, as a virtual box image file. It is preconfigured and contains all the needed software and datasets for the workshop.

To be able to start immediately we ask you to install VirtualBox on your computer already before the workshop. Here is how to do that:

1. Go to virtualbox.org and click the big Download VirtualBox 5.2 button.



2. On the next page select the VirtualBox Version suitable for your operating system. If your laptop runs Windows select the Windows host version and so on...

- **VirtualBox 5.2.0 platform packages.** The binaries are released under the terms of the GPL version 2.
 - [Windows hosts](#)
 - [OS X hosts](#)
 - [Linux distributions](#)
 - [Solaris hosts](#)

Update: The Guest Additions image with the 5.2.0 release fails to work with recent Linux guest kernels. Please try [this image](#) which we believe fixes the problem. For experimental Linux 4.14 support, please try [this image](#). Both should work on all older guest systems too: please file a report on the [bugtracker](#) if they do not.

3. Once the download is finished install VirtualBox on your computer.

You can familiarize yourself with VirtualBox by reading the manual which can be found also on virtualbox.org. The direct link is here:

<http://download.virtualbox.org/virtualbox/5.2.0/UserManual.pdf>

Especially the First Steps part contains some interesting information. We won't go into much detail for VirtualBox and we will also not create any new virtual machines.

You can try to download the original BioLinux image yourself and see if it works.

1. Navigate to <http://environmentalomics.org/bio-linux-download/>
2. Click on Bio-Linux Download in the menu on the left
3. Select the OVA file for use with VirtualBox/VMWare:

OVA file for use with VirtualBox/VMWare

>> [Download from main site](#)


4. Once the file is downloaded you can follow the instructions below on how to load the .ova image with VirtualBox.

This is a good exercise to familiarize yourself with VirtualBox and BioLinux and if the original image runs fine on your laptop the image provided by us will also run. However it is important to note that the BioLinux Image we provide on the first day of the workshop will differ in several respects from the original one. It will include additional software and contain several datasets we will analyze during the course.

On the first day of the workshop:

On the first day of the workshop we will provide you with a ready to use virtual machine image running BioLinux which you can load using VirtualBox.

Once you have downloaded the BioLinux image, this is how you load it into the VirtualBox:

1. Start VirtualBox.
2. Click on File -> Import Appliance
3. Click on  and select the genomics-workshop.ova image and proceed to the next step.
4. In the next Window you have the possibility to change some of the parameters of your VirtualBox image. You may want to change the amount of RAM available to your image depending on the amount of RAM you have available on your computer. The default is 2GB and the amount of RAM you assign should not exceed 50% of the RAM on your host computer.
5. Click import. **Importing may take some time!**
6. On the left side of the VirtualBox window you will now see an entry called genomics_workshop.
7. Select this entry and click Start (the big green arrow) or double click the entry to start the Virtual Machine.
8. If everything has worked you should shortly be seeing an new window similar to this. If you try this with the original BioLinux image the screen will look different.



9. This is the log in screen of your biolinux VirtualMachine. The user we are going to use is called **lichen symbiont** and the password for this user is **genomics**.

Here a few recommendations on the minimum hardware requirements:

CPU with at least 2-cores not older than 3 years

4GB of RAM

20GB of free disk space

2. Day 1 - 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. You will also learn to run shell and python scripts because the shell

includes a powerful scripting language (BASH). 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 BioLinux which is a very popular distribution based on Ubuntu Linux (Debian) 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 them yourself, experimenting as much as you can.**

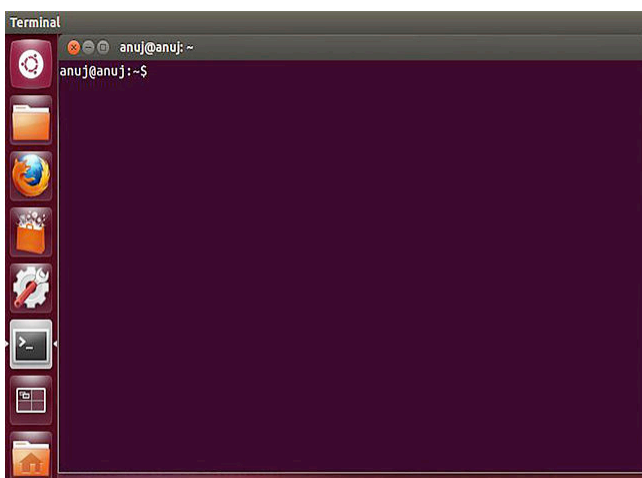


Figure: The terminal is an application, you can launch it from the applications menu, dock, or the Ctrl+Alt+T shortcut.

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 `symbiont@lichengenomics:~$` 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).

To execute commands in the shell you type your command and hit “Enter” (Return) to execute the command. The output will appear in the same window below the command prompt.

2.1 Navigating the file system

The first thing to do is to learn the basics of moving between directories 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 or a file’.

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/symbiont
```

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 genomics_course
$ cd data
$ pwd
/home/symbiont/genomics_course/data
```

You can see that the `/` symbol denotes levels of directories, so that ‘data’ directory is contained within the ‘genomics_course’ directory which is within the ‘symbiont’ 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/symbiont/genomics_course/
```

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/symbiont
```

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

The tilde symbol (`~`) is shorthand for this home directory so `/home/symbiont/genomics_course` and `~/genomics_course` 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/symbiont`

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 ~/genomics_course/data
$ pwd
/home/symbiont/genomics_course/data
$ cd
$ pwd
/home/symbiont
$ cd -
$ pwd
/home/symbiont/genomics_course/data
$ cd -
$ pwd
/home/symbiont
```

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 type operating systems 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/symbiont/genomics_course/data/day-1`.

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 again 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 raw data folder.

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.

```
$ ls
```

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 '`day-1`' and investigate what is present- a single [fasta sequence](#) file (`.fas`). You can do this of course by changing directory, print your working directory to check where you are, and listing the files present:

```
$ cd day-1
$ pwd
/home/symbiont/genomics_course/day-1
$ ls
scaffold.fas
```

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 dayy-1
```



```
bash: cd: dayy-1: 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 `genomics_course/raw_data/` 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 `scaffold.fas` file in `nano` you should use the general unix approach of `program-name filename`, and assuming we are still in the `day-1` directory:

```
$ nano scaffold.fas
```

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

TASK: To practice `nano` rename the sequence to `>fungal_scaffold` Save the changes using `Ctrl-O` (called writing-Out), it will ask you if you want to save to the same file, don't, give a new informative name like:

`scaffold_renamed.fas`

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 "parmelia_sequences.fas" which can be found in `~/raw_data/fasta/`. 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` (global regular expression parser).

`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 RPB1 in our `parmelia_sequences.fas` file we do the following:

```
$ grep "RPB1" 18S_parmelia_sequences.fas
```

This returns all the lines containing the word **RPB1**.

We can count how many of the sequences are RPB1 by using the *count* flag (`-c`) with `grep` as follows:

```
$ grep -c "RPB1" 18S_parmelia_sequences.fas
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 `scaffold.fas` and `parmelia_sequences.fas` 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 `RPB1` with `RPB_1` in our `parmelia_sequences.fas` file we could do the following:

```
$ sed 's/RPB1/RPB_1/' < parmelia_sequences.fas > RPB1toRPB_1.fas
```

This will replace the single word `RPB1` we identified using `grep` with the word `RPB_1`, but output these changes to the file `RPB1toRPB_1.fas`, 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 `parmelia_sequences.fas` file and outputting (`>`) to `RPB1toRPB_1.fas`.

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
```

instead of

```
sequences_2.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 (concatenate) 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. Do you remember that `<` determined the input source? Can you think of any situations in your work where this UNIX command line approach could save an enormous amount of work in manipulating files?

`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 such as having a defined number of columns or lines 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 `day-1/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.txt` 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. 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 `RNAtoDNA.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 `transcripts.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
```

First figure out how to run the script to produce a DNA file. Next, google how to rename files at the unix command line, and rename the newly created file to “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

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

In our **day-1** directory we have a shell script: **readmap_all.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. It is to give you an idea that

By doing a **cat readmap_all.sh** we can view the content of this file on screen.

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 **cd trimmed_reads** which changes the directory to trimmed_reads. It then immediately executes **files=\$(ls renamed_*R1*)**. This lists all the files which have renamed_*R1* in their filenames and saves the list of filenames to the variable named files. The stars sign (*) is a placeholder. It means any character(s). Next there is another cd command which makes the script jump back to the previous directory. The command **echo “Building index file:”** outputs this message to the screen. The next line is another comment. The command **bowtie2-build**
~/data/peltigera/Peltigera_membranacea/additional_data/Pmem_fungal_scaffolds_DNA_cleaned-up/Pmem_mycobiont_scaffold_1line.fasta
./02_bowtie/Pmem_fungal_index.build calls the program bowtie2-build. The bowtie2 software is used to align (map) short sequences, usually reads to much longer sequences such as assemblies. This process is called read mapping. The purpose of this can be manifold. For example read mapping is needed if you want to detect SNPs or look at expression level differences in RNASeq experiments. Of course there are many other applications for read mapping.

experiments. Of course there are many other applications in which read mapping plays an important role. Read mapping with bowtie involves several steps. In the first step bowtie2-build takes a reference genome against which short reads will later be mapped. In the process it creates an index file containing information used by the actual bowtie2 aligner. The line `echo "building index done"` will inform us that the index file was successfully built. The next two lines create two additional variables: `samfile_base=pmem_readmap` and `counter=0`. The `samfile_base` variable is used as a prefix for the new file which will be created during the rest of the shell script. You may have already guessed that this script was written to automatize readmapping for many sequenced libraries against one reference genome. The actual mapping takes place in the next few lines. Since we want to do this many times we use a for loop. This loop will repeat the commands between the `do` and `done` statement for a specific number of times. In this case it is done for the number of files in the `$files` variable which we created earlier. Inside the for loop, the first thing which is done here is to assign several new variables containing the file paths which should be mapped. Can you guess what the `sed` command in the script does? The file names `$d` and `$d2` are then displayed on screen with `echo`. In the following line the counter variable is increased by 1: `((++counter))`. After that a new variable is defined. Can you guess what this variable may contain?

The next command calls bowtie2 to map the reads in the read files against the reference:

```
bowtie2 -p 24 -q --phred33 --fr -x
./02_bowtie/Pmem_fungal_index.build -1 $name1 -2 $name2 -S
./02_bowtie/$samfile.sam
```

It uses many of the variables created earlier, so hopefully now it gets more clear why we need them. The next command is another `echo` command which tells us that the bowtie command is finished. bowtie2 creates a so-called SAM file which contains information about the mapped reads such as the name of the read, the sequence and the mapping coordinates. SAM files are text format files and you could theoretically look at them with `cat` although this may not be a good idea because SAM files can get very large. This is why we have to compress them into a compressed binary format. Compressed SAM files are called BAM files and we compress SAM files like this: `samtools view -bS ./02_bowtie/$samfile.sam | samtools sort - ./02_bowtie/$samfile` and work only with the compressed files. The next line only has the `done` command which indicates the end of the for loop. After the loop has finished the last few lines of code will combine all just created BAM files into a single file sort the file and create an index file.

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 at once. Entering all these commands with the correct flags and file locations from the command line directly would be very difficult, error prone and would take a lot of time 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 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.

2.6 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 tef1 sequences are there in the `parmelia_sequences.fas`?
2. What is the last sequence record? (No scrolling down please!)
3. Search for beta-tubulin and replace it with Btub
4. Demonstrate that you can (a) edit the file directly (b) save the edit as a new file

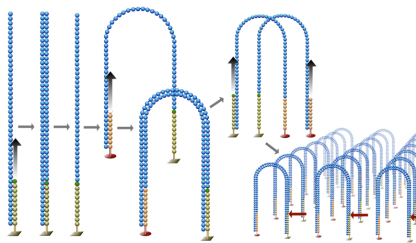
2.7 Some reading if you wish to extend your knowledge

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>

3. Day 2 – Analysis of High throughput sequencing data and genome assembly

3.1 A quick introduction to short read sequencing data

The *massively parallel* generation of nucleotide sequence data has had an enormous effect on modern biology, and is commonly referred to as *high-throughput sequencing* or also Next generation sequencing (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 genomic sequencing (among many other applications) 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 basic short read assembly of raw

NGS data.

Figure: Illumina data generation. See [Illumina sequencing](#) videos on YouTube.

We have provided sample data for you which is included in your VirtualBox Image. To find it, navigate to `day-2/illumina/`.

3.1.1 The FASTQ format

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. Phred scores are coded as single characters in ASCII format.

determine the number of sequences in a file using your command line skills? Extra points if you manage to do it without decompressing the data file to disk (hint: 'pipe')

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

Before we move on to quality assessment and quality trimming/filtering of Illumina data:

Task: Find out what the difference between 'phred-33' and 'phred-64' fastq sequence quality encoding is.

3.2 First steps with short read data

3.2.1 Basic quality control

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

```
$ fastqc
```

The program opens and you can directly load any (compressed) Illumina data file. 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](#) (<http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/>). The data has been produced on a HiSeq sequencing instrument. What do you think about your data in terms of 'per base sequence quality'? - Compare to example reports [here](#) (<http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>) - would you say the data quality is acceptable? Make sure you directly compare forward and reverse read files. What do you notice?

3.2.2 Read trimming with FASTX-toolkit

Now, we will perform basic quality filtering/trimming, using a set of simple but powerful programs provided in the [FASTX-toolkit](#) (http://hannonlab.cshl.edu/fastx_toolkit/). A full list of the available tools can be found [here](#) (http://hannonlab.cshl.edu/fastx_toolkit/commandline.html) with some usage instructions.

The FASTX tools are installed in the VirtualBox image. Try to call one of them in the terminal, e.g.

```
$ fastx_reverse_complement -v.
```

As usual, adding `-h` to the command displays some usage instructions. 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 trimming

Find the appropriate FASTX tool to perform basic quality trimming.

End-trim your forward reads to a quality of $Q > 30$ and discard all reads shorter than 100bp. How many reads were retained.

Task: Quality filtering

Find the appropriate tool and discard any read that has a quality score of $Q < 30$ across more than 40% of its length.

Please explore the effects this trimming procedures had on your data using FastQC.

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.

3.2.3 Advanced read trimming with trimmomatic

One other popular quality trimming software is [Trimmomatic](http://www.usadellab.org/cms/?page=trimmomatic) (<http://www.usadellab.org/cms/?page=trimmomatic>).

Hint: The jar file is located in /usr/local/bin/

One of the great things about trimmomatic is that it does ‘paired-end aware’ trimming, i.e. one can feed pe reads into a single operation and after trimming intact pairs and potential ‘orphan reads’ will be reported separately. **What might ‘orphan reads’ be?**

Here is an example command of Trimmomatic. Try to understand the different parameters by googeling the manual.

```
java -jar /usr/bin/trimmomatic-0.36.jar PE -phred33 -trimlog  
trimmomatic.log genomic_1.fastq genomic_2.fastq  
genomic_trimmomatic_paired_1.fastq  
genomic_trimmomatic_orphan_1.fastq  
genomic_trimmomatic_paired_2.fastq  
genomic_trimmomatic_orphan_2.fastq LEADING:30 TRAILING:30  
SLIDINGWINDOW:5:20 MINLEN:90
```

Task: Adjust the above command to fit your own data and trim all three datasets to a minimum read length of 100bp. Of course you are also free to try all kinds of different settings.

Break :)

3.3 Assembly

3.3.1 contigs, k-mers and De Bruijn graphs

Once we are satisfied with the trimmed data quality, we can now start to create longer sequences out of the short FASTQ reads. The resulting longer sequences are called contigs and the process of creating these contigs is called assembly. Sequence assembly is a complex problem and short read assembly can be computationally intense, especially when it comes to memory usage. Because there are many different kinds of assemblers for this purpose, (almost) all claiming to be better than the others but in reality each assembler has different pros and cons. A list of some commonly used assemblers is provided at the end of this chapter. At a certain stage of the assembly process most contemporary short read assemblers use a concept called De Bruijn graphs. A de Bruijn graph is a

method to represent a large number of short sequences in terms of their k-mer content in the form of graph structure. k-mers are all possible sequences (read fragments) of length k. The graph is constructed by aligning these k-mers so that the overlap between two k-mers is k-1 (see figure below).

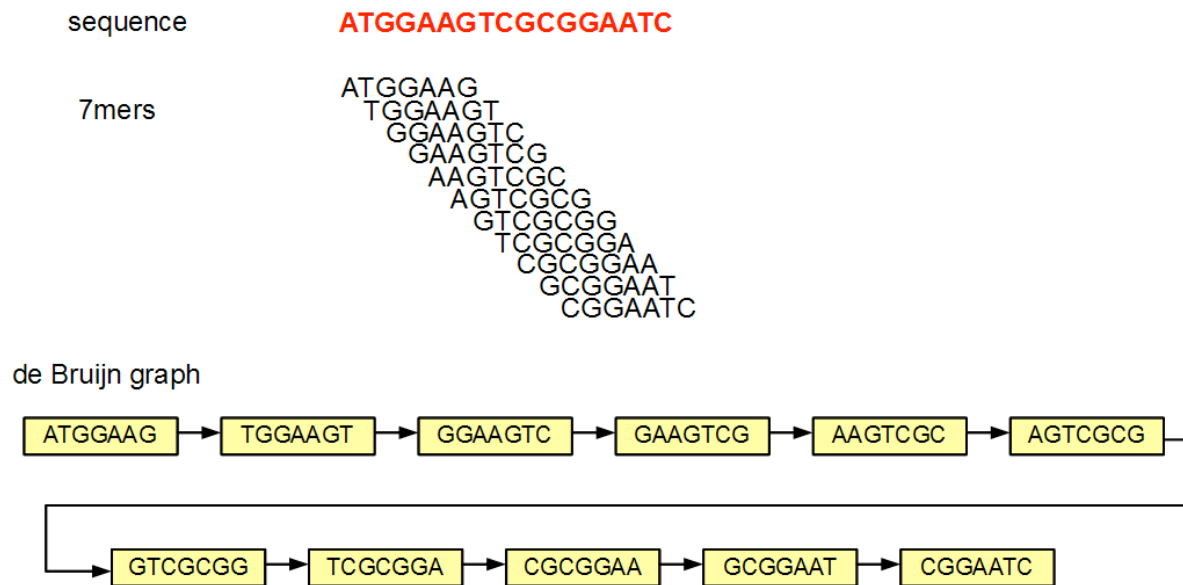


Figure: A simple De Bruijn graph without branching constructed of 7-mers.

Task: Can you think of possible roles the size of k can play in the assembly? What is the maximum size of k?

Check out this page for additional information on how k-mer size may influence assembly: <https://github.com/rrwick/Bandage/wiki/Effect-of-kmer-size>. The assembly graphs on this site are created with bandage (<https://rrwick.github.io/Bandage/>) an assembly graph viewer.

3.3.2 Assemblers

As mentioned previously there are many choices for assemblers (list at the end of the chapter). For the sake of this tutorial we will use minia (<http://minia.genouest.org>) and spades (<http://bioinf.spbau.ru/en/spades>) but you can try any assembler you want. Each assembler has different qualities and problems. We will now assemble the test datasets with different assemblers, it is up to you to choose which one you want to use, but please try at least two different ones with different parameters to see how the choice of software and parameters can influence assembly.

Tip: Of course you can also install any other assembler in your VirtualBox image and try to assembly the test data.

First lets create a new directory for the assemblies. Starting from your home directory type:

```
$ cd genomics_course/day-2/
```

```
$ mkdir assembly
```

```
$ cd assembly
```

It may also be useful to create a separate directory in your assembly folder for each assembly so that you know which file belong to which assembly.

3.3.3 Assembly with Minia

First lets run Minia because it is very fast and we will have results very quickly. Minia uses a filtering algorithm to make storing de Bruijn graphs more memory efficient. With Minia it is possible to assembly the human genome on a desktop computer within a day. A typical Minia command looks like this:

```
$ minia -in reads.fa -kmer-size 31 -abundance-min 3 -out output_prefix
```

The `-in` flag specifies the fastq read file from which an assembly should be made. minia accepts both plain FASTQ and gzipped FASTQ files. `-kmer-size` sets the used k-mer size. `-abundance-min` tells minia how often a k-mer must be seen in the reads to be considered correct. `-out` simple specifies the prefix for the output files.

Task: In your assembly folder, run minia to assemble the `illumina_reads_1_R1.fq.gz` file. Think of a good naming scheme for your different assemblies by changing the output prefix or by using. You will create several assemblies so a good naming scheme is important.

Once minia is finished you can run `ls` to see the files created by minia. The file `<yourprefix>.contigs.fa` is the assembly created by minia in FASTA format. Each sequence in the file represents one contig of your assembly. **Congratulations, you have just assembled your first genome!**

3.3.4. Assembly quality control with QUAST

Assemblies can differ in many aspects. For example the used assembler and the used parameters such as k-mer size but also read-quality, read length and whether one uses paired-end data will significantly affect assembly. It is therefore always important to compare different assemblies.

To check the quality of your assembly manually you can of course look at the contig file and count the length and number of the sequences to see how fragmented your assembly is. However there are several tools which will do this for you and provide you with many additional metrics to compare multiple assemblies.

For this we use quast (<http://quast.sourceforge.net/quast>), a python-based program with many features to evaluate the quality of assemblies. For full reference of all the options available for quast please refer to its manual: <http://quast.bioinf.spbau.ru/manual.html>.

To create a quast report you have to feed quast your newly created assembly. Run:

```
$ quast.py your_assembly.contigs.fa
```

This will create a new directory in your current folder call `quast_results`. This folder may

contain multiple additional folders (when you first run quast.py there will be two). Type `ls -l` to see them (an example is given below). The first folder called latest is a so-called **symlink**. Symlinks are files which point to other places in your file system. In this case latest points always to the latest quast report you created (for example `latest -> results_2017_10_22_11_58_21`).

```
total 16
lrwxrwxrwx 1 symbiont symbiont 27 Oct 22 11:58 latest -> results_2017_10_22_11_58_21
drwxrwxr-x 4 symbiont symbiont 4096 Oct 22 10:48 results_2017_10_22_10_48_45
drwxrwxr-x 4 symbiont symbiont 4096 Oct 22 10:49 results_2017_10_22_10_49_01
drwxrwxr-x 4 symbiont symbiont 4096 Oct 22 10:49 results_2017_10_22_10_49_10
drwxrwxr-x 4 symbiont symbiont 4096 Oct 22 11:58 results_2017_10_22_11_58_21
```

This is practical because you don't have to remember the new folder name created by quast, you can simply enter the latest directory to see your latest report.

Quast creates reports in many different formats, even graphically. The simplest way is to look at the file report.txt with cat.

```
symbiont@lichengenomics:~/genomics_course/day-2/assembly/minia/quast_results/latest$ cat report.txt
All statistics are based on contigs of size >= 500 bp, unless otherwise noted (e.g., "# contigs (>= 0 bp)" and "Total length (>= 0 bp)" include all contigs).

Assembly      assembly100.contigs  assembly1.contigs  assembly5.contigs
# contigs (>= 0 bp)  198                82                 12
# contigs (>= 1000 bp)  6                  32                 7
# contigs (>= 5000 bp)  0                  0                  5
# contigs (>= 10000 bp) 0                  0                  1
# contigs (>= 25000 bp) 0                  0                  0
# contigs (>= 50000 bp) 0                  0                  0
Total length (>= 0 bp)  73020              82629              78162
Total length (>= 1000 bp) 7309              60042              76351
Total length (>= 5000 bp) 0                  0                  70525
Total length (>= 10000 bp) 0                  0                  65082
Total length (>= 25000 bp) 0                  0                  26424
Total length (>= 50000 bp) 0                  0                  0
# contigs          37                 49                 8
Largest contig      1666              3964              26424
Total length        29369             72532             77240
GC (%)              30.55             28.74             28.95
N50                 812               1660              14512
N75                 657              1219              10632
L50                 15               15                2
L75                 25               27                4
# N's per 100 kbp  0.00             0.00             0.00
```

Figure: This is an example of a quast run comparing three different assemblies.

3.3.5 Quality measures of assemblies: N50 and L50

Do you know what all the different measures mean? Especially important are the N50, N75, L50 and L75 values. These are often used in publications and give a first hint about how good your assembly is in terms of continuity. Given a set of contigs with different lengths, imagine you would order them from longest to shortest. Starting from the largest contig take all the contigs you need, so that the length of these contigs comprises 50% of your total assembly length. The length of the shortest of those contigs is your N50. L50 is then simply the number of contigs you need to get your N50. N75 and L75 are similar but refer to 75% of your complete assembly. Look here for additional info on

N50: <http://www.molecular ecologist.com/2017/03/whats-n50/>

What would you think is an optimal N50?

Task: Take some time to explore the different results quast provides. You can also google the quast manual to find additional information. What is the N50 of your first assembly? How many contigs do you have?

3.3.6 Assembly with spades

Although minia is nice because it is quick and uses only very little memory it also has problems. For example it does not work with paired-end data and it only uses single k-mer sizes. Since we have paired-end data with forward and reverse reads we will use a different assembler named spades. Spades is more versatile and it works with multiple different read types including paired-end data. It is also able to utilize multiple k-mer sizes at once and merges them to create the best possible assembly. Spades has many more options, which are documented in its manual (<http://spades.bioinf.spbau.ru/release3.9.0/manual.html>). Most of them we will not use for this example. You can call spades like this:

```
$ spades.py -k 21 --pe1-1 illumina_1_R1.fq.gz --pe1-2  
illumina_1_R2.fq.gz -o ./spades
```

This is a very basic spades command using only very few of the available options. Lets look at them quickly: The first option `-k 21` specifies the k-mer sizes which should be used. In this case 21-mers. The next arguments specify the paired-end input files: `--pe1-1` for the first read pair and `--pe1-2` for the second read pair. `-o` specifies the output directory.

Note: Depending on the amount of memory (RAM) you have on your computer and the amount of RAM you set for your linux virtualbox image it may be a good idea to run spades with a memory cut-off `-m 1` so that the run does not crash because it allocates too much memory.

Spades creates many different output files. The most important ones are `contigs.fasta`, which contains the assembly, `assembly_graph.fastq` contains the assembly graph which can be viewed with bandage and some log files with information about your spades run (`params.txt`, `spades.log` and `warnings.log`).

Task: Look at your assembly graph files by starting Bandage with:

```
$ Bandage
```

Bandage is a program with a Graphical User Interface (GUI) and it is basically self explanatory. Also look at the wiki page for additional information: <https://github.com/rrwick/Bandage/wiki>. From the assembly graph can you determine the largest contig.

Hint: Spades also works with multiple k-mer sizes. To tell it to summarize the assembly based on several values of k change the according parameter like this: `-k 21,31` this will tell spades to use a k-mer size of 21 and 31.

3.3.7 The lichen genomics workshop assembler ton

Now everything is set up to do serious genome assembly:

Task: Assemble the test datasets with different assemblers using different parameters (k-mer sizes, paired and unpaired data, ...) and compare all of them with quast. What would you say is your best assembly? Please enter the stats of your assembly to this shared google docs spreadsheet for comparison:

https://docs.google.com/spreadsheets/d/1cReNkEemy87pAoDXXr48orfjXncLLs_WRcf0vAxlrBM/edit?usp=sharing

Advanced task: Create a shell script which runs multiple minia and spades assemblies. Put the output of each assembly into its own directory.

Advanced task: Install and assemble the reads with platanus.

There are many other ways to evaluate a genome assembly besides QUAST, for example you may be interested in the gene-completeness of your assembly. Although also quast has features to do this, there are better tools like BUSCO (<http://busco.ezlab.org>). We will use BUSCO in the next session, although in a slightly different context.

In general, if you consider an assembly good or bad largely depends on what you want to do with it. There is of course also a limit of assembly quality when you start with a single library. A genome assembly is usually only the very start of genomic analyses. Usually genomes with super high quality assemblies use a combination of libraries with different insert sizes, long reads and mate pairs.

If you are curious which genome you just assembled you can BLAST search your assembly at the NCBI database. What do you think it is?

3.4 If there is time: Read mapping

Now that you have tried different assemblers and have assembled some of the test datasets you may wonder what is next. Well, this largely depends on the aim of the respective question. For example in the next practical session we will take a closer look into phylogenomics. However there are many other possibilities. You may like to identify nucleotide differences of multiple different sequenced individuals or look at highly expressed genomic regions if you have additional RNASeq data available or you would simply like to know how often a certain stretch of DNA was sequenced in your library. All these tasks usually include a technique called read mapping or read alignment. It is the process of aligning large numbers of short reads produced generated in a high-throughput sequencing experiment to longer sequences (e.g. genome assemblies). And we will do just that now in this last exercise for today.

There are many different programs to do read mapping and they differ in their speed, accuracy and underlying mapping principles. We will be using bowtie2 which uses the Burrow's-Wheeler Transformation method, which is a method originally developed for data compression however the string matching functionality of the algorithm is also very useful in effectively handling short read data.

bowtie2 (and other aligners) create a so-called SAM (Sequence/Alignment Map) file which contains information about each mapped read such as the read name, mapping location, mapping quality

among others. The full specifications of SAM files can be found here: <http://samtools.github.io/hts-specs/SAMv1.pdf>.

We will now create a SAM file with bowtie2 by mapping the reads from our test-datasets against one of your assemblies. Do you remember in the last session we discussed a shell script to automatize read mapping with bowtie2 for many files. We will do exactly that however not by using a script but by doing every step independently.

3.4.1 Creating a bowtie index

Hint: You may want to create another folder to store all the files created during read mapping. It is up to you to decide where.

Like in the script the first practical session we first have to create an index file of the assembly which will be used by bowtie2 to align the reads. This is done with bowtie2-build which is called like this:

```
$ bowtie2-build path/to/assembly.fasta assembly_index.build
```

The first argument `bowtie2-build` needs is your assembly file. The second argument (assembly_index.build) is the prefix for all the mapping files which will be created by `bowtie2-build`. Change the command according to your needs. Creating the index should be quick and after it is done you may check the contents of your directory with `ls`. You can also check the newly created file with head however these files are in binary format and since we are not computers they appear to contain rubbish. bowtie2 does not use the original fasta alignment from this point on, it will only work with the binary index files.

Task: Create an indexed assembly with bowtie2

3.4.2 Read mapping with bowtie2

The next step is to actually align the short reads from the test datasets. Luckily bowtie2 accepts the FASTQ format as input (among others) so we are ready to go. There is a difference if you want to map unpaired or paired reads. For unpaired reads a bowtie2 command may look like this:

```
$ bowtie2 -p 2 -q --phred33 -x ./assembly_index.build -U  
/path/to/fastqfile.fq.gz -S samile.sam
```

And for paired read it may look like this:

```
$ bowtie2 -p 2 -q --phred33 -x ./assembly_index.build -1  
/path/to/fastqfile_1.fq.gz -2 /path/to/fastqfile_2.fq.gz -S  
samile.sam
```

OK, let us look at the different arguments bowtie2 uses. The `-p` flag tells `bowtie2` how many threads it should use. This means how many independent mapping processes it should start which will do the mapping simultaneously. On a powerful computer this is a good way to speed up the mapping process, especially if many (millions of) reads should be mapped. In our case `-p 2` is enough. The `-q` flag tells the program which kind of input format our reads have. In this case it is FASTQ. The `--phred33` you know already, it tells bowtie2 what quality encoding scheme should be used. `-x`

`./assembly_index.build` tells bowtie2 against which index it will map. Use the same prefix as in the bowtie2-build command. In unpaired mode bowtie2 only maps one FASTQ file to the assembly, this is specified with the `-U /path/to/fastqfile.fq.gz` command. If you want to map read pairs specify the forward and reverse read with `-1` and `-2` flags respectively. The last argument `-S samfile.sam` tells bowtie2 the name of the SAM file. This is our output file.

Hint: Call bowtie2 without any arguments to see all available options of the program listen on screen

Task: Now change the above commands according to your needs and perform read mapping

3.4.3 Save space by converting SAM to BAM

We could view the SAM file directly since they are plain text files (and you can try that if you want), however this is also the reason why they are usually very large and can quickly have several Gigabytes. Therefore it is a good idea to create a BAM file from the SAM file. A BAM file contains the same information however it is in binary format and thus much smaller than a SAM file. We will now convert the newly created SAM file to a BAM file using a program called samtools. Samtools (<https://github.com/samtools/samtools>) was developed specifically to handle and manipulate SAM and BAM files. In the process we will also sort and index the BAM file so that it can be read by the genome viewer we will use.

Task: To do this execute the following commands (make sure to change them according to your personal file paths):

```
$ samtools view -bS samfile.sam | samtools sort - bamfile.bam
```

```
$ samtools index bamfile.bam bamfile.bai
```

There are not many options here to discuss. In the first command the `-bS` flag tells samtools that the input file is a SAM file and the outfile should be a BAM file. The second part of this command, which is connected to the first with a pipe will sort the BAM file and save it to the specified location (here `bamfile.bam`). The next file uses the sorted bamfile and create a BAM file index with the `.bai` extension.

3.4.4 One way to look at a BAM file

We are now ready to view the BAM file with artemis. Artemis is a free light-weight genome viewer created by the Sanger Institute: <http://www.sanger.ac.uk/science/tools/artemis>. You can start it by running:

```
$ artemis
```

Hint: it is probably a good idea to do this in a new terminal window.

In the new window click on File -> Open and select the FASTA assembly file you used in the beginning of the readmapping exercise and click Open. The assembly will open in a new window. In this window click File-> Read BAM / VCF ... select and open the just created BAM file. Your artemis window should now look something like the figure below:

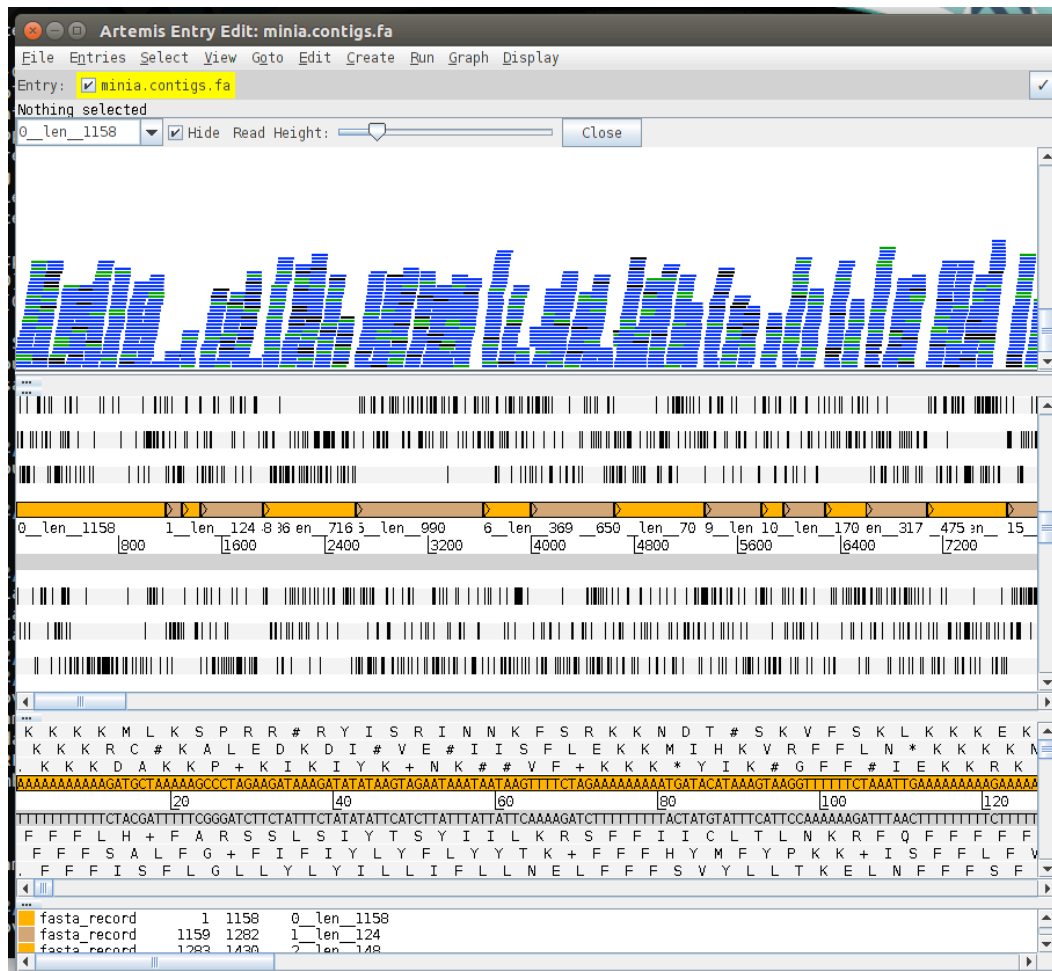


Figure: Artemis with a loaded assembly and corresponding read mapping.

Task: Explore your read-mapped assembly for a bit. Here is a link to the manual:

ftp://ftp.sanger.ac.uk/pub/project/pathogens/workshops/BioLinux_Artemis_Data/Artemis-BioLinux.pdf

What do you see? Can you give a rough estimate of sequence coverage in your assembly?

3.5 List of some common short read assemblers

Minia - <http://minia.genouest.org>

Spades - <http://cab.spbu.ru/software/spades/>

Velvet - <https://www.ebi.ac.uk/~zerbino/velvet/>

MaSuRCA - <http://www.genome.umd.edu/masurca.html>

Platanus - <http://platanus.bio.titech.ac.jp>

MIRA - <https://sourceforge.net/p/mira-assembler/wiki/Home/>

MITObim - <https://github.com/chrishah/MITObim>

AbySS - <http://www.bcgsc.ca/platform/bioinfo/software/abyss> (already installed on the virtual machine)

ALLPATHS-LG - <http://software.broadinstitute.org/allpaths-lg/blog/>

4. Day 4 –A simple phylogenomic pipeline

*“Hi, I downloaded the genomes you told me and finally managed to assemble the ones we sequenced last year... I find this is a great idea but... **How do I align the genomes?...**”*

4.1 Introduction

A common goal to all phylogenomic surveys is to provide a reconstruction of the evolutionary history of a taxonomic group using genome-wide data. In some cases reconstructing a phylogeny may be the ultimate goal of the experiment, but more often phylogenies provide a framework to interpret population dynamics, speciation events or provide the background signal to interpret comparative genomic trends.

The protocols commonly used in phylogenetic surveys to select, sanger-sequence, align and estimate a phylogenetic tree with a set of neutral loci are very standardized. In phylogenomics, however there is no single approach or consensus protocol that can be invoked as a consensus. In fact, due to human and computational limitations, analyzing a high number of loci requires a series of simplifications and compromises that largely depend on the type of sequencing platform, of genomic coverage and the purpose of the survey. The philosophy and methods used for every step are extremely dependent on the type of data acquired and the purpose of the survey. For instance highly refined Bayesian methods of model testing, coestimation of phylogeny and population parameters or even making simple phylogenetic inferences are not available for all types of data and often they do not scale well to cope with genomic datasets extremely limiting their use.

In this practical session we provide a succinct introduction to different methods used to generate phylogenomic data matrices, and focus on one of them, making use of orthology based gene sets which provides a simple and straightforward method that to address multiple background questions and provides a good entry-level approach to phylogenomics. The practical parts are highlighted.

4.2 Assembling a phylogenomic data matrix

The student question quoted above on the top of the page may seem naïve, but reflects the first problem one encounters when attempting to reconstruct a phylogeny from a set of genomic samples. As of today, several handbooks provide a well digested guide to phylogenomics, and several pipelines have been published or made available online. Few years back, the student from the quote would have started by googleing “phylogenomics” and “genome alignment” and would have spent a couple of days trying to figure out how to use the output of MAUVE [1], before assuming that having the contigs systematically sorted [2] is nice, but that it was not the point at all.

However, it remains difficult to make well-informed decisions on the design of a phylogenomic survey. First, there is an important terminological gap between the disciplines of phylogenetic systematics and molecular genomics which is often difficult to bridge; the term phylogenomics itself is widely used in the literature in surveys using evolutionary methods of genome annotation CITE and not just extended phylogenetic surveys. Second, there is no single method to assemble a genome-wide phylogenetic data matrix, starting for the genome representation of the library itself. This generates an important deal of noise, as highly impacting genomic surveys often deal with taxonomic groups other than fungi or algae, and as our neighbouring colleagues working on animals of plants usually engage in well-informed but highly opinionated digressions about their next project during coffee breaks feeding us up with sometimes inadequate information. Third, to complicate it even more, each manuscript uses a slightly different set of scripts or pipelines to automatize the assembly of a data-matrix. Pipelines that are often developed with a particular target organism in mind, bacteria, animals, humans, fungi... and are not always transferable to other organisms, especially lichens, where lichenized-fungi, their photobionts, and all the complex biotic community forming the lichen system provide advantages and disadvantages on their own.

4.4.1 DNA alignment based resequencing pipelines

The simplest simple way to assemble a data-matrix is to use an alignment-based genotyping approach. Such methods start by mapping/aligning a set of raw reads to a reference assembly using a short-read alignment tool as BWA [3] or Bowtie [4]. The resulting alignments are later processed using a variant calling algorithm to retrieve a set of SNP loci and/or haplotypes per sample; various methods of variant calling are found in most general purpose bioinformatic packages as GATK [5] – *CombineGVCFs*, *GenotypeGVCFs*–, samtools [6] –*mpileup*– and other.

The usability of short-read alignment methods depends on the degree of similarity between the query and target reference genomes. In general raw-read methods are most suitable for population surveys or for questions at very fine taxonomic scales, e.g. subgenera. They are widely used to analyze reduced representation libraries such as RNASeq [7], RADSeq/GBS –Stacks [8] and Pyrad [9] – or to process poolseq [10] libraries. It is worth mentioning that in some cases, the lack of a reference genome is overcome by assembling a transient reference library stacking short reads [11], a case that is of limited usability in interpreting lichen metagenomic samples.

The data matrices obtained with such methods are often more suitable in the framework of statistical population genetics, but in the case of haploid organisms they may also be processed to

use in standard tree-building frameworks. A usable phylogenomic pipeline using an alignment to reference approach is RealPhy [12] which was used for *Rhizoplaca melanophthalma* [13].

In general such experiments provide affordable setups to analyze many samples in reduced representation or resequencing experiments. Its drawbacks are the difficulty to incorporate the evolutionary history of individual genes into the equation, making stringent Heterozygosity-based filtering necessary to ensure some confidence in the “orthologous” relationship between the reference regions and the mapped reads.

4.1.2 Annotation based pipelines and Orthology

A second philosophy uses the alignment between protein sequences to assemble phylogenomic data matrices. These generally require that raw reads are assembled into query genomic draft from which genes are predicted and compared to a repository of protein sequences or evolutionary protein models.

Protein sequences are more conserved than their DNA counterparts, because of the redundancy of the genetic code and the constraints placed by functional and structural regions. These methods, profit from the methodological background of phylogenomics in the original sense (98), focused on functional annotation. For instance Hidden Markov Models (HMM) provide exceptionally flexible statistical descriptors of protein evolution.

These methods provide an improved framework to assemble phylogenomic data-matrices at deeper phylogenetic scales. They allow the assessment of orthology, to control for differences in copy number between genomes, and to identify putatively syntenic blocks, from which also intergenic spacers could be used as highly-variable neutral markers if desired [14].

While the concepts of orthology and paralogy are well-established in systematics [15], the complexity of Eukaryotic genomes and comparative genomic scenarios, required the introduction of new concepts and methods to interpret gene orthology. Paralogs can be in- and out-paralogs denoting that the genes were duplicated subsequent or prior to speciation, respectively. In recent duplications, in-paralogs may show co-orthology relationships with multiple genes, which complicates the solid detections of one-to-one orthologs across genomes. Several strategies have been employed to distinguish probable orthologs from paralogs. The simplest approach would be to use reciprocal best alignment hits [16] to inform about gene orthology. In lichens this method may provide reasonably good results at shallow phylogenetic scales, as generic and family levels where orthologous proteins are widely recognizable. This method requires an important level of *a posteriori* supervision and is clearly superseded by more complex methods [17], based on phylogenies [18] or of hmm profiles of proteins [19].

A straight forward approach to assemble orthology aware phylogenomic data matrices makes use of precompiled repositories of orthologous gene sets, as OrthoMCL [20], OrthoDB [21] or the underused fungal specific FUNYBASE [21]. Evolutionary pathways of protein evolution can be statistically summarized using probabilistic models called profile hidden Markov models (profile HMMs) [22], which provide very usable models combined to alignment-like search algorithms as HMMER3 [23].

Such methods provide a good tool to assemble phylogenetic data-matrices covering closely related organisms but also distant groups of organisms. Reference databases and HMM profiles must be comprehensive and fine-tuned for each particular phylogenetic scale. This can be laborious but makes the approach highly scalable to different phylogenetic scales. In this respect the software BUSCO v 3.0 [24], intended to estimate gene-completeness of genome assemblies, provides taxon specific sets of single-copy orthologous genes usable at different phylogenetic extents. The number of genomes included and their relatedness greatly influences the number of target single-copy orthologs inferred and modeled. For instance the BUSCO set for Eukaryotes contains 303 orthologs, while the one for Fungi contains 291, for Dykaria 1312, for Ascomycota 1315 and for Pezizomycotina a total of 3156 orthologs.

The development of focus sets of orthologs, even including positional orthology (synteny) in the equation are clearly the simplest and more robust resource to produce phylogenomic datasets for a wider range of experiments and focal groups. For this reason this approach is the one we will succinctly develop in the following toy pipeline

4.2 A phylogenomic pipeline

The somewhat naïve phylogenomic pipeline we propose makes use of the computational strategies implemented in the BUSCO v 3.0 [24] pipeline. BUSCO uses a repository of single copy orthologous genes defined by precompiled HMM profiles [22] to estimate gene-completeness of a genome assembly. Because BUSCO uses augustus [25] for gene-prediction and HMMER [23] for the identification of orthologs from a precompiled hmmer profile database, it covers three necessary steps in the development of a phylogenomic pipeline. A more thorough pipeline would profit from a) using a more thorough iterative gene-prediction strategy, b) using a custom built taxon specific ortholog database and probably a c) a further use of a phylogenetically explicit method of ortholog detection. A similar setup profiting from the use of BUSCO has been used in a large phylogenomic survey in *Saccharomycotina* [26] Fig. 1.

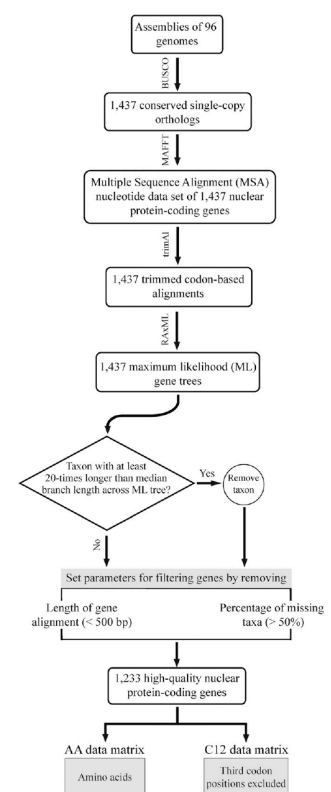


Figure 1: Pipeline used by Sheen et al. 2017 [26].

4.2.1 Assessment of gene-completeness with BUSCO

Task 1: Assess the gene completeness of a small genomic subset using busco.

1. Go to the folder made for today's activities `$ cd /genomics_course/day-4/`
2. Locate the busco pipeline in `./busco` and the downloaded lineage specific databases in `./busco/lineages`.
3. Decompress the lineage databases using `tar -xvzf` on each tar.gz file.
4. Look for the folder containing .hmm profiles: a) How many BUSCOs does each database

contain? b) Try to understand it the .hmm files by looking at it with **more** or a similar tool.
NOTE: It shows a table with the probabilities for each aminoacid state for each position in the sequence.

5. **Let's run BUSCO:** Select one of the reduced assemblies found in **day-4/task_1**
4. Run BUSCO taking care to specify the correct path for the python script as well as for your input file, the lineages and where you want the output to be.

```
$ python ../busco/scripts/run_BUSCO.py \  
-i YOUR_SEQUENCE_FILE.fa \  
-o OUTPUT_NAME \  
-l ../busco/lineages/NAME_OF_LINEAGE -m geno
```

5. Look at the output files. Several of them could be of use in phylogenomic pipeline, especially the aminoacid and nucleotide fasta files provided in the folder **/single_copy_busco_sequences** which would be easy to use downstream. Another file to take into account is the one named **training_set_*** which could be a good starting point for further gene-prediction and annotation pipelines. We will use the file named **full_table_*** downstream. It is both complete and intuitive, and serves multiple purposes that will be explained in task_2.
6. Now... a) Are the assemblies complete? b) Are there a lot of duplicated BUSCOs? c) What could be the cause? d) Could Kmer-coverage suggest the presence of more fungi in the sample? e) Does the assembly require additional cleaning? f) Can this generate a problem downstream? g) Why is the parameter length smaller than the value of Start-End. The following table provides a simplified example.

#BUSCO_group	Status	Scaffold	Start	End	Bitscore	Length
BUSCOIEOG7CCC0Z	Complete	NODE_8_length_770894_cov_4.06215_ID_270532	553762	589411	1175.1	909
BUSCOIEOG73RBMM	Complete	NODE_8_length_770894_cov_4.06215_ID_270532	641463	652863	933.3	530
BUSCOIEOG7DVDN4	Complete	NODE_63_length_278098_cov_4.11132_ID_673671	94607	106102	713.1	448

Task 1b: Asses the gene completeness of a genomic draft using busco.

For the highly motivated among you, if your VM or local computer is working fast enough try repeating the same steps with a complete genomic draft of the ones found in folder **./task_1b**. The files are tar-gzipped. You will first have to unpack them using the **tar -zxvf** command.

- a) How complete are they? b) Do they have a lot of duplicated BUSCOS? c) Do you observe notable differences in kmer-coverage between the scaffolds included in the genomic draft? again What does it mean? d) Does the assembly need further sanitation?

4.2.2 Parsing the BUSCO output for phylogenomics

Task_2: We developed a very simple R script to mine the output of BUSCO in order to produce a phylogenomic data matrix. In a well atomated pipeline it may be more reasonable to work with the

fasta files per busco and genome provided by the program. We do however use the `full_table_*` file for several reasons. First in a preliminary survey, working with the full_table file allows us to a) use BUSCO in the sanitation of a preliminary assembly, in addition to a blobology-like [26] approach using augustus, diamond [27] and MEGAN [28], b) thus to manually exclude one of the duplicated BUSCOS that may be causing problems for preliminary exploration. Also, we are not departing from well-tested nor evidence-based protein predictions, so using scaffold coordinates allows us to c) include intronic regions, and d) modulate possible missinterpreted regions in the alignment step. Finally, it also serves to provide a simple example on how to work with tables and sequences in R.

To simplify the rest of the pipeline we provide a reduced full_table file and assembly for four *Caloplaca* specimens and part of the *Xanthoria parietina* genome as a reference.

1. Go to folder `../task_2`. Find the files to use, they are named coherently as x1-x4.fas and .txt
2. Make the following directories (with `mkdir`) `./fastas`, `./alignments`, and `./trees`
3. Process the files in R, append sequences to individual files per BUSCO. Notice the simple for loop.

```
for i in x1 x2 x3 x4 xanpa;
do
Rscript parse_busco.r ./input_files/${i}.txt \
./input_files/${i}.fasta $i ./fastas/;
done
```

4. At this stage we add a completion filter to include only BUSCOS present in all 5 samples. This is not necessary, and there is a tendency to include missing data in phylogenomic matrices as a better practice. However, BUSCOS found in too few samples and not present in the outgroup should be systematically excluded. Subsequently align each individual fasta file with mafft

```
for FILE in ./fastas/*.*;
do
value=$(wc -l $FILE | cut -d' ' -f1)
if [ $value -eq 10 ] # HERE THE FILTER
then
echo $value
echo $FILE
mafft --retree 2 --maxiterate 2 --adjustdirection --
thread 2 $FILE > $PWD/alignments/${FILE##*/}
else
mv $FILE ./fastas/out/$FILE
fi
done
```

5. **Not today...** At this stage we may want to refine the alignment using muscle, and potentially produce multiple alignments with different methods to calculate a consensus alignment or to use further in the `-compareset` option of trimal.
6. Then we will trim the alignment using the software trimAl [29]. You can find suggestions and a tutorial in <http://trimal.cgenomics.org>. First explore the alignment report for your files using the `-sgt` and `-sident` flags. An example to create a report file could be

```
mkdir ./trimal_reports
for FILE in $PWD/alignments/*.*;
do
trimal -in $FILE -sgt >> ./trimal_reports/report.txt;
done
```

7. Then trim the alignment using an automated procedure, check the webpage and tutorials for details. We save an html alignment report and a phylip alignment file

```
mkdir ./refined
for FILE in $PWD/alignments/*.*;
do
s=${FILE##*/}
s=${s%.*}
trimal -in $FILE -automated1 -htmlout
$PWD/trimal_reports/${s}.html
trimal -in $FILE -automated1 -phylip >
$PWD/refined/${s}.phy
done
```

8. Almost over we calculate single gene trees using raxML, do not forget to a) rename the labels that were reversed (look at the files mafft called some sequences_R_x...) and b) root the trees.

```
for FILE in $PWD/refined/*.phy
do
sed -i -e 's/^_R_/g' $FILE
done
```

OK solved....

```
mkdir ./trees
for FILE in $PWD/refined/*.phy
do
NAME=${FILE##*/}
NAME=${NAME%.*}
raxml -s $FILE -n $NAME -x 12345 -p 23456 -f a -# 100 -m
GTRGAMMA -T 4 -o xanpa
mv ./RAxML_* ./trees/
done
```

9. Finally we use RaxML to calculate a majority rule consensus tree which we annotate with the Internode Certainty and Tree Certainty scores (IC, ICA, TC, TCA) proposed by Salichos and Rokas [30–32]. The resulting consensus tree can be visualized in the program dendroscope [33].

```
#Concatenate all gene-trees into a single file
cat RAxML_bipartitionsBranchLabels.* > all_trees.tre
```

```
#Calculate MR consensus and annotate  
raxmlHPC -L MR -z all_trees.tre -m GTRCAT -n -T1
```

Additional task: Take a look at section X. Of the RaxML manual. Are there more options to calculate a consensus and to annotate multiple trees?

Additional task: Try to program two similar steps using IQtree [34] instead of RaxML, it can be slightly faster [35] and it incorporates automated model-testing, which is a very interesting addition.

Additional task: Now try to wrap up all the latter steps into a single sequential script, pack it into a .sh file and try running it as a pipeline.

4.2.4 Single gene trees and consensus

It has become obvious that having a multiplicity of genes does not only provide information as a consensus for the whole genome. Different regions of the genome may have different histories and a consensus may not conform to a simplified ditichotomous structure as provided by a phylogenetic tree. A great tool to explore the phylogenetic signal contained at a whole genome level is the software Dendroscope 3 [33], which provides a wide ranges of methods to estimate rooted networks for the further exploration of the phylogenetic signal encountered across loci. An additional program useful to estimate evolutionary networks accounting for incomplete lineage sorting is Phylonet [36], ecceTERA [37] provides an interesting method for *a posteriori* reconciliation of genetrees and species trees.

Additional Task_3: Contains an additional set of 964 gene trees calculated from the same Caloplaca dataset. a) Use RaxML to summarize them, b) open `dendroscope` and import the provided datafile. Explore the algorithms provided to to compute rooted networks, consensus and hybridization networks.

4.3 References

1. Darling AE, Mau B, Perna NT. Progressivemaue: Multiple genome alignment with gene gain, loss and rearrangement. PLoS One. 2010;5.
2. Rissman AI, Mau B, Biehl BS, Darling AE, Glasner JD, Perna NT. Reordering contigs of draft genomes using the Mauve Aligner. Bioinformatics. 2009;25: 2071–2073.
3. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics. 2009;25: 1754–1760.
4. Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Methods. 2012;9: 357–9.
5. McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytksy A, et al. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res. 2010; 1297–1303.
6. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence

- Alignment/Map format and SAMtools. *Bioinformatics*. 2009;25: 2078–2079.
7. De Wit P, Pespeni MH, Ladner JT, Barshis DJ, Seneca F, Jaris H, et al. The simple fool's guide to population genomics via RNA-Seq: an introduction to high-throughput sequencing data analysis. *Mol Ecol Resour*. 2012;12: 1058–1067.
 8. Catchen JM. Stacks: an analysis tool set for population genomics. *Mol Ecol*. 2013;22: 3124–3140.
 9. Eaton D a R. PyRAD: assembly of de novo RADseq loci for phylogenetic analyses. *Bioinformatics*. 2014;30: 1844–1849.
 10. Schlötterer C, Tobler R, Kofler R, Nolte V. Sequencing pools of individuals — mining genome-wide polymorphism data without big funding. *Nat Rev Genet*. Nature Publishing Group; 2014;15: 749–763.
 11. Catchen JM, Amores A, Hohenlohe P, Cresko W, Postlethwait JH. Stacks : Building and Genotyping Loci De Novo From Short-Read Sequences. *G3 Genes|Genomes|Genetics*. 2011;1: 171–182.
 12. Biozentrum Universität Basel. RealPhy 1.12 [Internet]. [cited 1 Jan 2017]. Available: <https://realphy.unibas.ch/fcgi/realphy>
 13. Leavitt SD, Grewe F, Widhalm T, Muggia L, Wray B, Lumbsch HT. Resolving evolutionary relationships in lichen-forming fungi using diverse phylogenomic datasets and analytical approaches. *Sci Rep*. Nature Publishing Group; 2016;6: 22262.
 14. Magain N, Miadlikowska J, Mueller O, Gajdeczka M, Truong C, Salamov AA, et al. Conserved genomic collinearity as a source of broadly applicable, fast evolving, markers to resolve species complexes: A case study using the lichen-forming genus *Peltigera* section *Polydactylon*. *Mol Phylogenet Evol*. 2017;
 15. Fitch WM. Distinguishing Homologous from Analogous Proteins. *Syst Zool*. 1970;19: 99.
 16. Ward N, Moreno-Hagelsieb G. Quickly finding orthologs as reciprocal best hits with BLAT, LAST, and UBLAST: How much do we miss? *PLoS One*. 2014;9: 1–6.
 17. Chen F, Mackey AJ, Vermunt JK, Roos DS. Assessing performance of orthology detection strategies applied to eukaryotic genomes. *PLoS One*. 2007;2.
 18. Kuzniar A, van Ham RCHJ, Pongor S, Leunissen JAM. The quest for orthologs: finding the corresponding gene across genomes. *Trends Genet*. 2008;24: 539–551.
 19. Sonnhammer ELL, Eddy SR, Birney E, Bateman A, Durbin R. Pfam: Multiple sequence alignments and HMM-profiles of protein domains. *Nucleic Acids Res*. 1998;26: 320–322.
 20. Li L, Stoeckert CJJ, Roos DS. OrthoMCL: Identification of Ortholog Groups for Eukaryotic Genomes -- Li et al. 13 (9): 2178 -- *Genome Research*. *Genome Res*. 2003;13: 2178–2189.
 21. Zdobnov EM, Tegenfeldt F, Kuznetsov D, Waterhouse RM, Simao FA, Ioannidis P, et al. OrthoDB v9.1: Cataloging evolutionary and functional annotations for animal, fungal, plant, archaeal, bacterial and viral orthologs. *Nucleic Acids Res*. 2017;45: D744–D749.
 22. Eddy SR. Profile hidden Markov models. *Bioinformatics*. 1998;14: 755–763.
 23. Mistry J, Finn RD, Eddy SR, Bateman A, Punta M. Challenges in homology search: HMMER3

- and convergent evolution of coiled-coil regions. *Nucleic Acids Res.* 2013;41: 1–10.
24. Simão FA, Waterhouse RM, Ioannidis P, Kriventseva E V., Zdobnov EM. BUSCO: Assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics.* 2015;31: 3210–3212.
 25. Stanke M, Schöffmann O, Morgenstern B, Waack S. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics.* 2006;7: 62.
 26. Shen X-X, Zhou X, Kominek J, Kurtzman CP, Hittinger CT, Rokas A. Reconstructing the Backbone of the Saccharomycotina Yeast Phylogeny Using Genome-Scale Data. *G3 Genes|Genomes|Genetics.* 2016;6: 3927–3939.
 27. Buchfink B, Xie C, Huson DH. Fast and sensitive protein alignment using DIAMOND. *Nat Meth.* Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2015;12: 59–60. Available: <http://dx.doi.org/10.1038/nmeth.3176>
 28. Huson D, Mitra S, Ruscheweyh H. Integrative analysis of environmental sequences using MEGAN4. *Genome Res.* 2011;21: 1552–1560.
 29. Capella-Gutiérrez S, Silla-Martínez JM, Gabaldón T. trimAl: A tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics.* 2009;25: 1972–1973.
 30. Kobert K, Salichos L, Rokas A, Stamatakis A. Computing the Internode Certainty and related measures from partial gene trees. *bioRxiv.* 2015; 22053.
 31. Salichos L, Stamatakis A, Rokas A. Novel information theory-based measures for quantifying incongruence among phylogenetic trees. *Mol Biol Evol.* 2014;31: 1261–1271.
 32. Salichos L, Rokas A. Inferring ancient divergences requires genes with strong phylogenetic signals. *Nature.* Nature Publishing Group; 2013;497: 327–331.
 33. Huson DH, Scornavacca C. Dendroscope 3: An Interactive Tool for Rooted Phylogenetic Trees and Networks. *Syst Biol.* 2012;61: 1061–1067.
 34. Minh BQ, Anh Thi Nguyen M, von Haeseler A. Ultra-Fast Approximation for Phylogenetic Bootstrap. *Mol Biol Evol.* 2013;30: 1188–1195.
 35. Zhou X, Shen X, Hittinger CT, Rokas A. Evaluating fast maximum likelihood-based phylogenetic programs using empirical phylogenomic data. *bioRxiv.* 2017;<http://dx>.
 36. Than C, Ruths D, Nakhleh L. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinformatics.* 2008;9: 322.
 37. Jacox E, Chauve C, Szöllosi GJ, Ponty Y, Scornavacca C. EcceTERA: Comprehensive gene tree-species tree reconciliation using parsimony. *Bioinformatics.* 2016;32: 2056–2058.