Exercise 2 - Genome download & overview – how & what software?

**Day 1**

## Conda/mamba setup:

Make sure you have your conda environment set up. Without any exceptions, everything in these exercises are installed with the conda framework. If you start googling your way into ‘sudo apt-get’ commands, call for help.

Then make sure you have mamba installed:

**conda install mamba -n base -c conda-forge**

Now you can use any command that previously used ‘conda’ with mamba, making it much faster

Remember that we install things with this syntax (which is not a real command, just the syntax!):

**mamba install PACKAGE -c REPOSITORY**

PACKAGE is here the package you want and REPOSITORY is what database it is in. In our case, it is almost always ‘bioconda’ and sometimes also ‘conda-forge’. The homepage/github of the software will usually tell you.

## Package installation:

Let’s start by installing the program ‘seqtk’. We do that like this:

**mamba install -c bioconda seqtk**

## Environments

Now we create a specific environment for the next step. This keeps programs from interfering with each other. This environment will be called ncbiDL, because we will download from NCBI from here. You can call it whatever you want though, doesn’t matter, but certainly helps with a logical name. Here, we use the command ‘create’ with mamba and we then use the ‘-n’ argument (short for ‘name’) to specify the name.

**mamba create -n ncbiDL**

This is then activate like this:

**mamba activate ncbiDL**

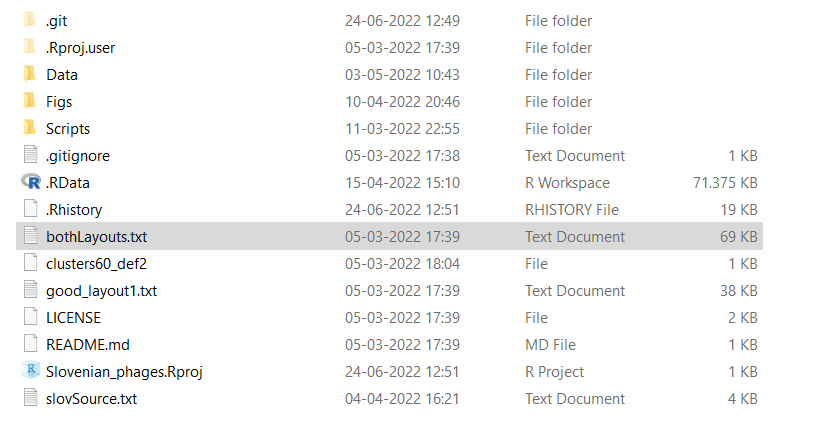
## Finding packages

Next, we need to find a good program to download genomes from NCBI. As we often do, we will google our way forward – try by googling ‘ncbi genome download’ and see if there is a good github repository that looks like it might fit (Hint: it should be the first hit on google by a nice user called Kai Blinn). Follow the install instructions and remember we use conda for installation (Second hint: exchange the ‘conda’ with ‘mamba’). Much of bioinformatics is looking through githubs and working through installation guides.

## Project hygiene

As soon as you have the program installed, we will do some data hygiene. **First, make a folder for this part of the summer school and enter it**. Call it whatever you want, but make sure it still makes sense for you next month. Within that folder, consider making a folder for the raw input – a good name for this folder could be ‘input’ or ‘genomes’ or similar. In this case, the next program will actually make the folder, so not necessary here. Always consider the architecture of your folders within a project, because it very easily becomes very messy and you will 1) not be able to remember the details of what is what and 2) your collaborators will have absolutely no idea. Also, your supervisor doesn’t have redo all your analysis when you have left and the paper is back from review.

Commonly, my projects look like this, a couple of loose files, but otherwise organized into input (Data), Output (Figs), analysis code (Scripts) and some files that reminds me what this project is about (README.md).



## Genome download

Now we download. The ncbi-genome-download package is a wonderful program that makes downloading genomes very easy. Getting genomes manually from NCBI is almost impossible in large scale.

First, check the possible arguments of the program like this:

**ncbi-genome-download -h**

Using either ‘-h’ or ‘--help’ is standard for getting programs to tell us how they want to be used.

The following command will download some genomes of P. piscinae. Take a close look at the arguments I have specified and make sure you know what they mean. Why do I specify the ‘-F’ argument like that? What do I mean with the ‘-l’ argument? What is the ‘-p’ argument and should you use the same value?

**ncbi-genome-download --genera "Phaeobacter piscinae" -F fasta --flat-output -o genomes -l complete -p 10 bacteria**

## Checking the files

Now we should have a couple of files in the ‘genomes’ folder. Check it out with the command

ls -l genomes/

These files are gzipped, evident by their file extensions (.gz). Gzipped files, analogous to zipped files on windows, are compressed and takes up less space, but are not readable. We will gunzip these files by use of the gunzip command, which is built-in in Ubuntu.

This will be our first use of the wildcard expansion of bash, e.g. the symbol ‘\*’, which stands for everything. Run this

**gunzip genomes/\***

which basically means “gunzip everything in this folder”. Check again if the files are now unzipped fasta files.

## Installing non-standard software

Next we annotate. This gets a little tricky, since the program we will use is written in Perl, and Perl programs do not play nice with conda, which is mainly a python manager. Seems like the Perl language interpreter program ends up in the wrong place, unless you install before the actual program. The program is Prokka, written by Torsten Seemann who (despite his use of Perl) writes great and easy-to-use software. The actual program finds all the coding regions of the genome, and then tries to identify what these genes are. It is fairly normal that most genes have an unknown function.

We will have to make a new environment and do some workarounds. The program is described here <https://github.com/tseemann/prokka>

**Make a new environment and activate it**. **VERY IMPORTANT - do not just keep working in your previous environment!** A good name for the environment could be ‘prokka’ or ‘annotation’, your choice.

Install bioperl first:

**mamba install -c bioconda perl-bioperl**

then install prokka as specified on the github:

**mamba install -c conda-forge -c bioconda -c defaults prokka**

Check that the installation worked by simply asking the program for help as described previously.

## Annotating

Now we annotate all the genomes. We need to run prokka on each of the genomes, which we can reasonably do one-at-a-time at this scale. We will, however, use a loop for this to make things efficient.

This would be a good time to make an output folder to put all our annotations - ‘annotations’ sound like a reasonable naming choice. Prokka will make a new folder for each genome within this folder if it is specified like below. Make sure to check with the description of the arguments so you are sure what each argument means. Try for one of the genomes before trying the loops.

The basic command is (for a given genome file, which here is GENOME.fna, and a given number of CPUs, which here is 8). ‘GENOME’ is not the correct name, nor is the GENOME.fna a correct file - you need to point to the correct genome as well as use a proper name for the outdir and prefixes. If the genome file is GCF\_0000001.fna, then a proper name could be GCF\_0000001 and so on.

**prokka --compliant --centre UniLjub --cpus 8 --outdir annotations/GENOME --prefix GENOME GENOME.fna**

On my laptop, running one genome with a single core takes about 25 minutes. With 8 cpus, around 4.

## Loops

Loops are great for doing the same thing many times. It might be avoidable here, but if you need to do basically the same operation a million times, then you need the loops. They are very fundamental in programming, and you will not get far without.

The format of a bash loop is this, like you might know from other languages:

**for i in LIST; do**

**SOMETHING**

**done**

This can be run in one line, a so called one-liner (note the semicolons, ‘;’):

**for i in LIST; do SOMETHING; done**

Here, the LIST is any type of list that contains something to loop over – in this case, it will be the genomes. First, try this:

**for i in genomes/\*; do echo $i; done**

As we can see, this code will run through everything in the genomes/ folder thanks to the ‘\*’ symbol, and then use the $i variable to store each of them, and in this case, just print the name, one at a time.

Its obviously good to have the full path of the genome files, but we need only the actual genome name for naming output folders. We will expand this with a little bit of extra code to fish out only the genome name, which is now captured in the variable $j by use of the ‘basename’ function which chops of the directory and, in this case, also the specified file extension ‘.fna’:

**for i in genomes/\*; do**

**echo $i;**

**j=$(basename $i ".fna");**

**echo $j;**

**done**

or as a one-liner:

**for i in genomes/\*; do echo $i; j=$(basename $i ".fna"); echo $j; done**

Your assignment is now to modify the last one-liner to run all genomes through PROKKA. In the above loop, you have the full path to the genome given in $i, you have the name of the genome given in $j, and you know the prokka command from earlier. If you have a computer with many cores, have a look in the help or on the github on how to use multiple threads (hint: the --cpus argument). In any case, this will take a while on a normal laptop.

For the more advanced users with bigger machines, the job can be run with the parallel package like this

**ls -d genomes/\* | parallel -j CPUS ' echo {} ; echo {/.}'**

where CPUS is the number threads you want and {} is analogous to $i and {/.} is analogous to $j.

**Done for the day!**

**Day 2:**

## Phage prediction

First we will predict phages. Here we will use phispy. Have a look online on how to install this program (github is a good bet). As always, try to use the conda framework for installation (but again change the ‘conda’ part to ‘mamba’).

If you want a nice graphic interface, you can try uploading the genomes to PHASTER at <https://phaster.ca/> . It looks to me that phispy is the best algorithm, but PHASTER shows the results nicely although it is not useful for high throughput analysis.

Ask phispy how it likes to be run, and also look closely the github. Run the first genome and note that it would like to be fed a genbank-file (e.g. a file with the .gbk ending), which is a very standardized annotation file. Perhaps the annotation we did yesterday gave us one…?

Phispy can do a bunch of different outputs. Often, we are particularly interested in the fasta files of the predicted prophage, so see if you can get phispy to give you that.

Obviously the loop logic we used yesterday can be used again. Same logic – loop over .gbk files and place the output in a folder somewhere it makes sense. Only issue is the fact that the files are now in individual folders rather than nicely collected in one. Some solutions to this: the ‘find’ command can find files with a pattern within folders. Surely the way to find files with a given extension/format can be googled. This can then be used in a loop like so and treated just like before:

**for i in $( FIND\_COMMAND ); do echo $i; j=$(basename $i .gbk); echo $j; done**

Make sure you understand, you will need it later. Start thinking of the biological inferences here – how many phages do you see in each genome? Are they in the same position on the genome? Is it the same phages in all the genomes? How would you compare them?

## Prediction of Biosynthetic Gene Clusters

Now we do antismash, which is the current gold standard for finding gene clusters that encode secondary metabolites, also called Biosynthetic Gene Clusters (BGCs).

Once again, installation proves tricky, and the online guide is outdated. Here is the installation code (which might take a while):

**mamba create -n antismash**

**mamba activate antismash**

**mamba install -c bioconda hmmer2 hmmer diamond fasttree prodigal blast muscle glimmerhmm**

**mamba install -c bioconda -c conda-forge antismash=6**

**download-antismash-databases**

Running antismash is however easy enough:

**antismash GBK\_FILE**

Once again, ask antismash how I would like to be run and then write a nice loop that can do it. Be aware that antismash will use all your cores by default, so make sure to handle that appropriately.

Antismash gives a bunch of outputs which are a little difficult to parse automatically. Luckily, It also gives a html file that you can view in your browser. Here, we might be interested in the total number of secondary metabolites, what types they are and if there are some new ones in there that antismash doesn’t know. Usually, the most interesting ones are NRPSs, PKSs and RiPPs, which are traditionally the ones we use as antibiotics and various other important functions. Do you find any of these BGCs on the phage regions? Can you run antismash on the prophage files from the phispy analysis directly?

## Whole genome phylogeny

Lastly, we will do the phylogeny. Here we use roary to fish out all genes that are sufficiently common across our genomes to be compared. Then we align them and build the tree from that. In more detail, roary uses blast to compare every protein of every genome, and within a certain cutoff of similarity (95% by default), considers proteins to be core genes if they are present in most (99% by default) of genomes. The nucleotide sequence of these core proteins are then put back-to-back for each genome and this sequence is then aligned for all genomes. This is a clever and data-driven approach to compare whole genomes, as it includes all relevant DNA rather than single marker genes or MLST schemes. Choice of cutoffs are obviously of key importance.

**Roary** is a Perl program, so we might run into a couple of issues again and the online guide does not work.

Start by making a new environment, then install bioperl and then install roary in that order (again using conda/mamba) just like we did with prokka.

**mamba install -c bioconda roary**

If you still get a ‘Can't locate Bio/Roary/CommandLine/Roary.pm” error, tell bash which Perl version to use like this:

**export PERL5LIB=$CONDA\_PREFIX/lib/perl5/site\_perl/5.22.0/**

Once again, ask roary how it wants to be run. It specifically wants .gff file, which luckily is one of the outputs of prokka. Run it like this, and make sure you understand the details of this command. The ‘-p 8’ argument might not be correct for your machine. This will take some time in any case.

roary -e --mafft -p 8 -f roary\_out $(find annotations/ -name "\*.gff")

What does roary consider a core gene in this case? Is that a reasonable criteria?

## Phylogenetic trees

Roary made a simple tree based on the presence and absence of genes that you can see in the ‘accessory\_binary\_genes.fa.newick’ file (.newick is a standardized tree format). You can visualize this on the command line using ete3:

**mamba install -c bioconda -c conda-forge ete3**

**ete3 view -t accessory\_binary\_genes.fa.newick --text**

But you are probably better off looking at it with a dedicated tree viewer. For an online solution, ITol is pretty good <https://itol.embl.de/> . If you are comfortable in R, there are multiple ways to make beautiful trees such as <https://guangchuangyu.github.io/software/ggtree/> . CLC is also very good: <https://digitalinsights.qiagen.com/products-overview/discovery-insights-portfolio/analysis-and-visualization/qiagen-clc-genomics-workbench/>

Lastly, we make the actual alignment into a true phylogenetic tree. We will use FastTree, which is probably the only tree-builder that is fast enough for such a big alignment. Install fasttree with mamba, find out how it works and run it on the aligned DNA from roary (you want the file ‘core\_gene\_alignment.aln’ and you should also specify the -gtr argument, which is the probably most robust ‘generalized time reversible’ algorithm). This will also take a while.

Have a look at the tree with one of the options above.

## Free play

Next up is free play, see if you can annotate your tree with the information we got from the earlier analyses. **You are done when you have a beautiful tree**.

You can do loads more analysis than this, like ‘abricate’ if you are interested in antibiotic resistance? It is from the same author as prokka and likely has the same Perl issues, but can be installed directly into the previous prokka environment: <https://github.com/tseemann/abricate>

You can repeat the analysis with your favorite bacteria if you wrote all the code correctly! Feel free to find more interesting things to look for - regardless of what you find interesting, someone probably wrote some software for it. If not, maybe you are the one that should do it?