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

**Day 1**

1. **Running commands:**

As per the lecture, linux commands take this form:

**COMMAND --FULLFLAG ARGUMENT1 -b ARGUMENT2 -c**

An example:

**RibDif --genus Phaeobacter -i .99 -f**

This command is for RibDif to run on Phaeobacter with an identity threshold of 99% and use fragmented genomes. Specifically, RibDif is the command and then a **full argument flag** is specified with 2 dashes (--genus) and its corresponding argument (Phaeobacter), followed by a **short argument flag** specified with 1 dash (-i) followed by its argument (.99). The last **short flag** (-f) has no argument.

Note that most flags have a long and short version (--genus or -g), and some flags require no arguments. The syntax is very much dependent on the author and must be individually investigated.

## 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.

**? Can you explain to you person beside you what conda is?**

**? Why do we use mamba and what is the difference between mamba and conda?**

## Package installation:

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

**mamba install -c bioconda seqtk**

The program seqtk is now installed in your base environment. You can check if that is in fact the case by sampling running:

**seqtk**

Which should simply return a helpful set of commands on how seqtk can be run.

**? Without any googling, what does the program seqtk appear to be used for?**

## Environments

Now we create a specific environment for the next step. This keeps programs from interfering with each other, which is particularly important when it comes to versions of software which may not work with one another. Imagine package A (v1.2) requiring >1.5 of package B, but package C must have version 0.9 of package B – wont work. Instead, we have multiple different environments containing compatible packages for each purpose, and hence accept that we often have the same package installed multiple times.

This particular environment will be called ncbiDL, because we will **D**own**L**oad from **NCBI** using packages in this environment. You can call this environment whatever you want though, doesn’t matter, but it certainly helps with a logical name. Here, we use the command ‘create’ with mamba and we then use the ‘-n’ flag (short for ‘--name’) to specify the name.

**mamba create -n ncbiDL**

This is then activate like this:

**mamba activate ncbiDL**

**? Do we need an environment for every single package we install?**

**? Considering that you should specify software versions in your paper, can you find an additional advantage of the environment concept?**

## 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.

**? Are there other ways to install this package? Should we use this?**

**? How would you download these genomes without this package and how long would it take?**

## Project hygiene

As soon as you have the program installed, we will do some data hygiene. **First, make a folder (with ‘mkdir’) for this part of the summer school and enter it (with ‘cd’)**.

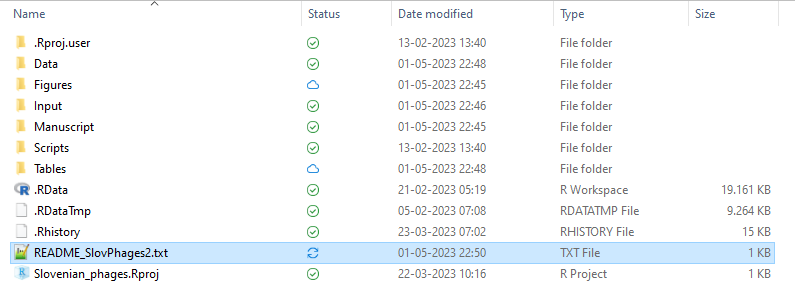
**mkdir MYFOLDER**

**cd MYFOLDER**

Call it whatever you want, but make sure it still makes sense for you next month when you have forgotten all you did in this course (‘MYFOLDER’ is a bad name). Within that folder, you should generally 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 which is problematic because   
1) you yourself will not be able to remember the details of what is what and   
2) your collaborators will have absolutely no idea what you did.   
3) Perhaps most importantly, your supervisor needs to make sense of your analysis, because they don’t want to redo all your analysis if/when you leave and the paper is back from review.

Commonly, my data-heavy projects look like this, which in this case is based on a Rstudio-project. Very few loose files for RStudio bookkeeping, but otherwise organized into raw data (Input), processed data (Data), Outputs (Figures & Tables), analysis code (Scripts) and a README that reminds me what this project is about (README\_SlovPhages2.txt). Since this is for a paper, there is also a Manuscript folder containing drafts.



**? What did you call your project and why?**

**? What do you think should go in your README file?**

## Genome download

In your case, you should not have anything in your folders yet. So, 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 without it.

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**

**? could you use the short version of the flag in some of the flags?**

**? why is there a ‘naked’ argument at the end without a corresponding flag?**

## Checking the files

Now we should have a couple of files in the ‘genomes’ folder. First, check out what you actually have now with

**ls -l**

Check out what is in your newly made ‘genome/’ folder 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 with the commands above.

**? although you can go in and out of folders to run commands where you actually are, what is the advantage of staying where you are and running commands from there?**

**? what is the meaning of the -l flag for ‘ls’? Try to combine the -l flag with -t, -S, -h, -r alone and in combination and see if you can guess what they do?**

## 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 it 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.

**! If prokka gives any other output than a list of commands, ask!**

## 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.

**? Can you successfully annotate a single genome?**

**? why do we use the –prefix flag?**

**? How many cores should you use?**

## 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 them.

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 first make a list of everything in the genomes/ folder thanks to the ‘\*’ wildcard symbol, and store each element of this list sequentially in the $i variable at each round of the loop. In this simple loop, the current value of $i is merely printed in each iteration of the loop.

It is generally good practice to use the full path of whichever files you work on (e.g. ‘~/genomes/GENOME1.fna’ rather than just ‘GENOME1.fna’) to ensure that you refer to the correct 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 full 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:**

Today, the technical instructions will be much more sparse since you now know how to both install & run programs, as well as how to do loops.

## Phage prediction

1. First we will predict phages using the program ‘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’).
2. 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.
3. Ask phispy how it likes to be run, and also look closely at 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…?
4. Phispy can do a bunch of different outputs. Apart from the coordinates of these phages, we are often particularly interested in the actual sequences of the predicted prophages, so see if you can get phispy to give you that in a .fasta file. Note that it uses a somewhat complex binary numbering system here.
5. Of course, the loop-logic we used yesterday can be used again when you are ready to run phispy on all your genomes. Same logic – loop over .gbk files and place the output in a folder somewhere it makes sense. The main challenge here, however, is the fact that the files are now in individual folders rather than nicely collected within one single folder. Luckily, this is such a common occurrence that Linux has a built-in function for it: the ‘find’ command, which finds files according to a pattern within folders (and much more). Surely the syntax for finding files with a given extension/format can be googled (such as .gbk-files). 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**

1. Make sure you understand, you will need it later. Start thinking of the biological inferences here before you get lost in all the data.

**? 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?**

**? Do you see the same phages as in PHASTER?**

## Prediction of Biosynthetic Gene Clusters

1. Now we run the antiSMASH algorithm on our genomes, which is the current gold standard for finding gene clusters that encode the production machinery to produce secondary metabolites, also called Biosynthetic Gene Clusters (BGCs).
2. Once again, installation might prove tricky, and the online guide may be outdated. Follow this installation code exactly (which might take a while):

**mamba create -n antismash antismash**

**mamba activate antismash**

**download-antismash-databases**

1. Running antismash is however easy enough:

**antismash GBK\_FILE**

1. 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. Also be aware that antiSMASH outputs a folder named by the input file unless you tell it otherwise.
2. AntiSMASH gives a bunch of outputs which are a little difficult to parse automatically. If you are comfortable with databases, the .json files contains the most information. For the rest of us, we can rely on the .html-files, which can be visualized in a standard browser. Navigate to the antiSMASH output-folder and use the following command to open the folder in windows and then simply inspect the html files in your browser of choice:

**explorer.exe .**

1. Here, we might be interested in the total number of BGCSs, which types they are and if we have any novel BGCs . Usually, the most interesting ones are NRPSs, PKSs and RiPPs, which are traditionally the ones we use as antibiotics and various other important functions.

**? What where the dominant classes of BGCs?**

**? Are there any BGCs common to all genomes?**

**? Can you check for BGCs on the phage regions?**

## Whole genome phylogeny

1. Lastly, we will do the phylogeny, that is, estimate how closely related our genomes are. 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 sequences 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 fairly clever data-driven approach to compare genomes, since it keeps as much information as possible given the data. In contrast, older methods rely on single genes, such as the 16S rRNA gene or specific collections phylogenetically specific genes, e.g. MLST.
2. **ROARY** is a Perl program, so we might run into a couple of issues again – we use a bit of shortcut here, which hopefully works:

**mamba install -c bioconda roary roary -y**

1. 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/**

1. This may or may not require 5.26 instead of 5.22; call the teacher in any case.

Once again, ask ROARY how it wants to be run. It specifically wants .gff files, which luckily is one of the outputs of prokka. Run it as below, 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, likely longer than you have time left in class – consider running it at home overnight if you have a small PC.

**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?**

**? would this work if we move up in phylogeny, such as families?**

**? inspect the ‘summary\_statistics.txt’ file – what does that tell you?**

**? What is in the ‘core\_gene\_alignment.aln’ file and why can we use it to build the tree?**

## 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 by installing and 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/>

For a real tree, though, we should 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, so consider doing it at home if your machine is slow.

Have a look at the tree with one of the options above. If your machine does not have the resources to build the tree, ask the teacher to get a pre-made version.

## 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 now 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?

Some suggestions for further analysis:

1. ABRICATE for antibiotic resistance profiling?
2. DBCAN for profiling of carbohydrate active enzymes (CAZYmes)?
3. PyANI for average nucleotide identity comparisons between genomes?
4. Customized BLAST-searches for your favorite gene?

**Feel free to as the teacher for input – I’m very happy to help and I’m only here for one week!**