Carleton Bioinformatics Documentation

Rika Anderson

Aug 21, 2020

Protocols:

CHAPTER 1

Week 1: Pre-lab software checklist

Rika Anderson, Carleton College

This bioinformatics class will, obviously, require us to do some hands-on bioinformatics. Most of the software you need is already installed on the remote server (i.e. a giant computer) that we'll learn how to log onto next week. However, you'll also need other software for some analyses. Usually, this software is installed in the Carleton computer labs where we do our in-person labs, but this year is obviously a little different.

If you are at Carleton, this software is available on the lab computers in CMC110 and in the Hulings computer lab. If you don't want to have to worry about software installation on your own computer, I recommend working in one of those two computer labs.

If you are working from home or working at Carleton with your own laptop, you will need to install software for this class using one of the two options below. Folks at ITS can help you with this if you run into problems.

Option 1: REMOTELAB. ITS has built a REMOTELAB environment for accessing lab software from personal computers. ITS Wiki

Option 2: Installation on your local computer. ITS has pulled together information on what software can be installed on personally owned computers. ITS Wiki

Software that you should have installed on your own computer:

1.1 Terminal to log on to a remote server

If you have a Mac, you should be all set.

If you have a PC or some other operating system, you should install a Ubuntu terminal if you can.

If that doesn't work, you can use PuTTY.

1.2 Text editor

If you have a Mac, I recommend BBEdit's free version.

If you have a PC, Microsoft Notepad works, and it should already be on your computer.

1.3 Seaview: alignment visualization

Available for Mac or PC. Link here

1.4 IGV Viewer: mapping and SNP visualization

Available for Mac or PC. There is also a web version that works similarly and may be fine for our purposes, but I haven't thoroughly tested it. Link here

CHAPTER 2

Week 2: Introduction to Unix/Linux and the Server; Assembly with IDBA-UD; ORF Calling and Annotation with Prokka

Rika Anderson, Carleton College

As you work through lab this week and for all subsequent weeks: Questions? Problems? Running into errors? Post on the lab Slack. I'll be monitoring the Slack during the weekly lab sessions (and outside of those times as well, though I can't always guarantee a rapid response.) It's easiest to answer your questions if you post a thorough explanation of exactly what command you ran and exactly what error message you're getting. Please feel free to answer others' questions too– some of you may run into the same problems.

2.1 Connecting to baross

2.1.1 1. About baross

We are going to do most of our computational work on a remote server (or computer) called baross, which is a remote server with 96 CPUs, 50 TB of storage, and 768 GB RAM. baross lives in the basement of the CMC. You can access it from lab computers on campus, and you can also access it from your own computer at home. First, you have to learn how to access baross.

If you are in a computer lab on the Carleton campus: Boot as a Mac user on the lab computer.

If you are working on a personal Mac computer: You should be able to follow the directions below.

If you are working on a personal computer that is operating Windows or another OS: You will need to find a way to connect to a remote server. I recommend installing a Ubuntu terminal. If that doesn't work, you can use PuTTY. (This was included in last week's pre-lab prep.)

2.1.2 2. Opening Terminal

If you're on a Mac, find and open the Terminal application (it should be in "Applications" in the folder called "Utilities"). If you're on a PC or other OS, open the window you'll use to ssh into a remote server (like Ubuntu or PuTTY or something similar).

The terminal is the interface you will use to log in to the remote server for the rest of the course. It is also the interface we will be using to run almost all of the bioinformatics software we will be learning in this course. You can navigate through files on the server or on your computer, copy, move, and create files, and run programs using the Unix commands we learned earlier this week.

2.1.3 3. ssh

We will use something called ssh, or a secure socket shell, to remotely connect to another computer (in this case it will our class server, baross). Type the following (substituting *username* with your own Carleton username– the same name as your email address):

```
ssh username@baross.its.carleton.edu
```

2.1.4 4. ssh (cont.)

You should see something like this: `[your username]@baross.its.carleton.edu's password:`

Type in your Carleton password. NOTE!! You will not see the cursor moving when you type your password. Never fear, it is still registering what you type. Be sure to use the correct capitalization and be wary of typos.

2.1.5 5. pwd

Whenever you ssh in to baross, you end up in your own home directory. Each of you has your own home directory on baross. To see where you are, print your current working directory.

```
# How to print your working directory (find out where you are in the system)
pwd
```

2.1.6 6. Your path

You should see something like this: `/Accounts/[your username]`

This tells you what your current location is within baross, also known as your **path**. But before we actually run any processes, we need to learn a few important things about using a remote shared server.

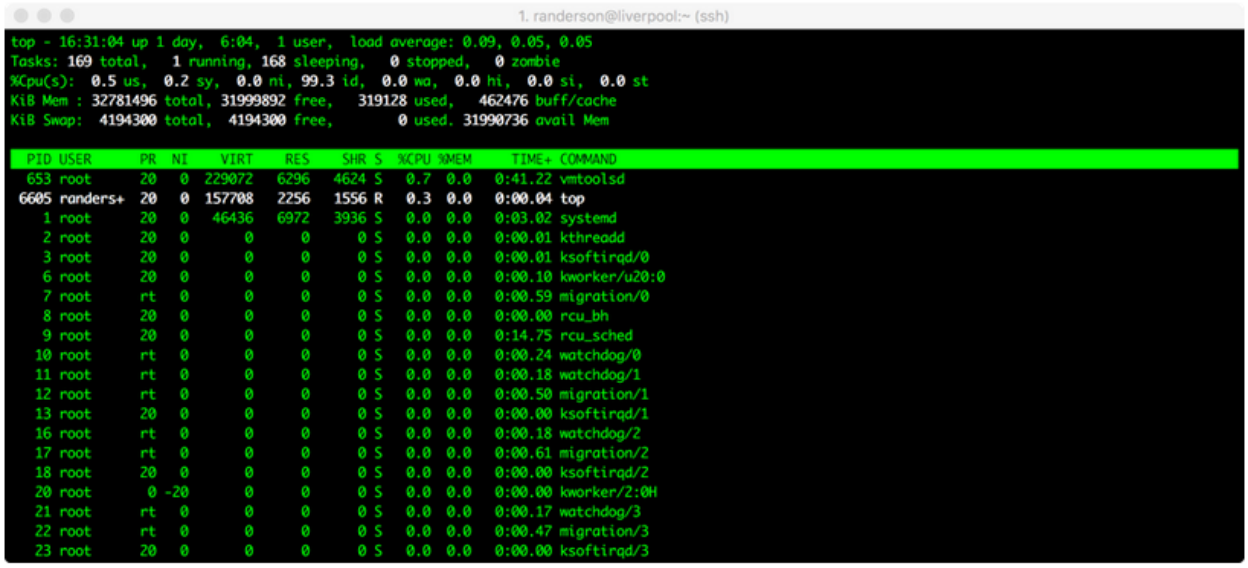
Important things to know about when using a remote shared server

2.1.7 7. top

One of the most important things to learn when using a remote shared server is how to use proper **server etiquette.** You don't want to be a computer hog and take up all of the available processing power. Nobody earns friends that way. Similarly, if it looks like someone else is running a computationally intensive process, you might want to wait until theirs is done before starting your own, because running two intensive processes at the same time might slow you both down.

To see what other processes are being run by other users of the same server, type: `top`

You should see something like this:



top screenshot

Here, we can see that randerson is the only user on baross, I am using a process called top, and it's taking up 0.3% of one central processing unit (CPU), and zero memory. Not much is going on in this case. (All the other things listed under 'root' are processes that the computer is running in the background.) If someone's process says 100.0 or 98.00 under %CPU, it means they're using almost one entire CPU. **baross has 96 CPUs and 768 gigabytes of RAM total.** This is also my research server, so please be courteous and leave some CPUs for my research students. A good rule of thumb is to try to leave at least 15 CPUs available at any given time. If we try to overload the computer, it means that everyone else's processes will have to be shared among the available CPUs, which will slow everyone down. For example, if it looks like 80 CPUs are already being used at a particular time, you might want to wait until later to run your own process.

To quit out of top, type: `q`

2.1.8 8. screen

Sometimes we will run processes that are so computationally demanding that they will run for several hours or overnight. Some processes can sometimes take several days, or even weeks. In order to run these processes, you want to be able to close out your session on the remote server without terminating your process. To do that, you use screen. Type: `screen -S test`

Your terminal will open a clean window. You are now in a screen session that you have called 'test'. Any processes you start while you are in a screen session will continue running even after you log off of the remote server.

Let's say you've typed the commands for a process, and now you're ready to exit your screen session to let it run while you do other things. To leave the screen session, type: `Control+A d`

This will "detach" you from the screen session and allow you to do other things. Your screen session is still running in the background.

To resume your screen session to check on things, type: `screen -r test`. (Now you've re-entered your screen session called 'test.')

To kill your screen session (this is important to tidy up and keep things neat when your process is finished!) type: `Control+A k`.

- The computer will say: `Really kill this window [y/n]`. You type: `y`.
- If this doesn't work, you can also type `exit` to terminate the screen session.

2.2 Creating an assembly and evaluating it (toy dataset)

2.2.1 9. mkdir

Let's say we've gone out in to the field and taken our samples, we've extracted the DNA, and we've sent them to a sequencer. Then we get back the raw sequencing reads. One of the first things we have to do is assemble them. To do that, we're going to use a software package called IDBA-UD. Bioinformaticians love to debate about which assembler is best, but ultimately it usually depends on the nature of your own dataset. If you find yourself with data like this someday and you want to know which assembler to use, my advice is to try a bunch of them and then compare the results using something like Quast, which we'll test below. For now, we're going to use IDBA-UD, which I've found to be a good general-purpose assembler.

Make a new directory in your home folder:

```
mkdir toy_dataset_directory
```

2.2.2 10. cd

Change directory into your toy dataset directory:

```
cd toy_dataset_directory
```

2.2.3 11. Copy toy dataset

Copy the toy dataset into your assembly directory. Don't forget the period at the end! This means you are copying into the directory you are currently in.

```
cp /usr/local/data/toy_datasets/toy_dataset_reads.fasta .
```

2.2.4 12. Run idba-ud on toy dataset

Run idba-ud on the toy dataset. Here is what these commands mean:

1. Invoke the program `idba-ud`
2. The `-r` gives it the "path" (directions) to the reads for your toy dataset. `../` means it is in the directory outside of the one you're in.
3. The `-o` flag tells the program that you want the output directory to be called "toy_assembly"

```
idba_ud -r toy_dataset_reads.fasta -o toy_assembly
```

2.2.5 13. cd to output directory

When that's done running, go into your new output directory and take a look at what's inside:

```
cd toy_assembly
ls
```

2.2.6 14. The output files

You should see several files, including these:

```
contig.fa
contig-20.fa
contig-40.fa
contig-60.fa
contig-80.fa
contig-100.fa
scaffold.fa
```

Next week, we will talk more about how assembly works. Briefly, De Bruijn graph assembly works by searching reads for overlapping words, or "kmers." IDBA-UD starts by searching for small kmers in the short reads to assemble those short reads into longer contigs. Then it uses those constructed contigs for a new round of assembly with longer kmers. Then it does the same with even longer kmers. It iterates through kmers of length 20, 40, 60, 80, and 100. You should get longer and longer contigs each time. Each "contig-" file gives the results of each iteration for a different kmer size. The "scaffolds.fa" file provides the final scaffolded assembly, which pulls contigs together using the paired-end data.

2.2.7 15. Examine output

Let's examine the assembly output files. First, take a look at your final scaffold output:

```
less scaffold.fa
```

2.2.8 16. Fasta file

You should see a fasta file with some very long sequences in it. When you're done looking at it, type: `q`.

2.2.9 17. Evaluate assembly quality

Now we're going to evaluate the quality of our assembly. One measure of assembly quality is N50, which is the contig length for which the collection of all contigs of that length or longer contains at least 50% of the sum of the lengths of all contigs. We could theoretically calculate the N50 by hand, but often times you'll get many, many contigs and it's very tedious to do by hand. So we will use an assembly evaluator called Quast. Run it on your toy dataset. (Note that the command is kind of long, so scroll right in the little green box below.)

```
quast.py contig-20.fa contig-40.fa contig-60.fa contig-80.fa contig-100.fa scaffold.
↪fa -o toy_assembly_quast_evaluation
```

Here is what the commands mean:

1. Invoke the program `quast.py`
2. We tell it to run the program on all of the output files of your toy assembly. Every fasta file that we list after `quast.py` will be included in the analysis.
3. The `-o` flag will create an output directory that we will call `toy_assembly_quast_evaluation`

2.2.10 18. View output by copying to your local computer

Quast gives you some nice visual outputs, but it's easier to look at these if you copy them from the server to your local computer. For this, we will use `scp`, or 'secure copy.' It's a lot like `cp`, except that you're copying from a remote server to your own computer. We will use `scp -r` because we're copying a directory recursively, not just a file.

1. Open up another Terminal window on your own computer. Navigate to your home directory.

```
cd ~
```

1. Type this (substituting your own username below):

```
scp -r username@baross.its.carleton.edu:~/toy_dataset_directory/toy_assembly_quast_
↪evaluation .
```

That means: copy something from the remote server baross and put it right here on my local computer. 3. In your newly copied folder, double-click on the file called "report.html" and take a look. It reports the number of contigs, the contig lengths, the N50, and other statistics for each of the assembly files you entered. The graph at the bottom shows the cumulative length of your assembly (y-axis) as you add up each contig in your assembly (x-axis). Think about what the variables mean, and how the trends change with each iteration of the assembly. We'll talk about this more in class, but for now, please keep a copy of the report.html somewhere and we'll refer back to it when we cover assembly in class next week.

2.2.11 19. Bookkeeping

For the sake of future bookkeeping, you'll need to modify your scaffold assembly file so that another software program we will use in the future (called `anvi'o`) is happy. (Spaces and other weird characters are so tricky sometimes.) **This will be very important for your project datasets as well.** Do this while you are inside the directory with your assembly files:

```
anvi-script-reformat-fasta scaffold.fa -o toy_dataset_assembled_reformatted.fa -l 0 --
↪simplify-names
```

2.3 Create and evaluate assembly (project dataset)

You're now going to run an assembly on your project dataset. Before we start working on your project datasets, I encourage you all to start a computational lab notebook. I recommend that you open a text document in BBEdit or a similar text editor and save it on the cloud somewhere. In that notebook, it will be crucial that you keep a record of all the commands you run, especially for your project datasets. That way, if you get a funny result, or if you want to repeat an analysis, you can always go back to your lab notebook to see exactly what command you ran. In my own computational lab notebook, I record the dates and I include notes to myself about the analysis, or observations of the graphs I make. I recommend using a text document (like in BBEdit) rather than Word or Google Docs because you can copy-paste Unix commands from a text editor without worrying about formatting. These notebooks are just for you– I won't be checking them– but now that we're about to get into the real analysis, you should start one for your project datasets and maintain them throughout the rest of this course.

2.3.1 20. screen

Since the assembly will take about 10-20 minutes, I recommend running this on screen. Type:

```
screen -S assembly
```

2.3.2 21. Make and change directories

Make a new directory in your home folder called "project_directory," and then change directory into your newly created directory:

```
cd ~
mkdir project_directory
cd project_directory
```

2.3.3 22. Copy project dataset

Next, copy your project dataset in to that directory. Your assigned project dataset is listed in an Excel file that is on Moodle.

For example, if you have dataset `ERR590988` from the Arabian Sea, you would do this:

```
cp /usr/local/data/Tara_datasets/Arabian_Sea/ERR598966_sample.fasta .
```

2.3.4 23. Start assembly

Next, start your assembly. **Substitute the name of your own sample dataset for the dataset shown here.**

```
idba_ud -r ERR598966_sample.fasta -o ERR598966_assembly
```

2.3.5 24. Detach from screen

After you've started the process, detach from screen:

```
Ctrl+A d
```

These assemblies will likely take some time. You can move on to the next steps ("Searching for and annotating open reading frames") while this runs, and then come back to this when it's done.

To check on the progress of your assemblies, you can either use `top` to see if your assembly is still running, or you can re-attach to your screen using the instructions above.

2.4 Searching for and annotating open reading frames

Now that we have assembled our contigs, we want to find genes on them. That requires identifying open reading frames (ORFs), and then comparing the ORF sequences with existing databases to see if we can figure out what kinds of genes they are. There are lots of programs to identify and annotate ORFs. We're going to use a program called Prokka, which wraps a bunch of different software into a pipeline that is nice and streamlined. Prokka basically does three things:

1. Identify ORFs
2. Compare those ORFs to databases of known genes to find the closest match and assign their putative functions
3. Several other things (like finding rRNA or tRNA genes, CRISPRs, etc.) that we aren't going to worry about right now.

Prokka works well and it's free, so we're using it today. It tends to work best for archaeal and bacterial genomes. If you want to identify and annotate ORFs for eukaryotic genomes, there's lots of similar software out there to do that.

Go to your toy dataset directory and make a new directory:

```
cd ~/toy_dataset_directory
mkdir ORF_finding
cd ORF_finding
```

2.4.1 25. Run Prokka

Now run Prokka on your toy assembly, which is located in the toy_assembly folder:

```
prokka ../toy_assembly/toy_dataset_assembly_reformatted.fa --outdir prokka_toy
```

This means you're invoking prokka on your reformatted toy dataset assembly, and you're putting it in a new directory called `prokka_toy`.

2.4.2 26. View output in FASTA format

You should see a directory called `prokka_toy`. Use `cd` to go into that folder, then use the program `less` to look at `PROKKA_09252018.faa` (or something like that– adjust according the date). You should see a fasta file with amino acid sequences. Each amino acid sequence is an open reading frame (ORF), or a putative gene that has been identified from your assembly.

The cool thing is that Prokka has also annotated your genes with their putative function. You can see that in each sequence title, which provides the name of the sequence assigned by Prokka (e.g. KGLPOPID_00002) and the putative function (e.g. Proine/betaine transporter). A lot of them will say `hypothetical protein`, which simply means that Prokka couldn't find a good match for that protein in public databases.

Note that the file that ends in `.ffn` contains the same thing, except the ORF sequences are in nucleotide format, not amino acid.

2.4.3 27. View output in Genbank (gbk) format

Prokka is handy because it gives you output in all kinds of formats, which you could use for other downstream applications. For example, it gives you a Genbank output, which is commonly used in the National Institute of Health (NIH) National Centers for Biotechnology Information (NCBI) database, which we'll be using a lot over the next few weeks. Take a look:

```
less PROKKA_09252018.gbk
```

For each contig, the Genbank file will give you information like the name of the contig, its length, the organism it comes from, and then information about every ORF or coding sequence (abbreviated as CDS) that contig. This includes the gene name, the location, the translation table (some organisms use slightly different codons and therefore different translation tables), its product, and its amino acid sequence. At the end of the list of ORFs for each contig, it gives you the DNA sequence of the contig.

2.4.4 28. View output in tab-separated column (tsv) format

Let's look at one last output format, which is in tab-separated columns, and therefore best visualized in a spreadsheet application like Excel. Use `scp` to copy this file from the server to your own computer. Remember, type this into a Terminal window that is NOT logged on to the server.

```
scp username@baross.its.carleton.edu:~/toy_dataset_directory/ORF_finding/prokka_toy/
↪PROKKA_09252018.tsv .
```

2.4.5 29. Open tsv file

Find your file on your local computer, and open your `.tsv` file in Excel (or Google Sheets). The column headers are in columns as follows, left to right:

1. `locus_tag`: the name that Prokka assigned the ORF
2. `ftype` (feature type): whether the thing it found is an ORF (CDS) or a tRNA or rRNA gene or something else.
3. `length_bp`: length of the gene
4. `gene`: if there was a match to an identified gene, it will give the name of that gene.
5. `EC_number`: the Enzyme Commission assigns numbers to known genes according to their function. If there was a match to a known gene with an EC number, that is given here.
6. `COG`: the Clusters of Orthologous Groups (COG) database groups proteins into categories of known function. If your ORF matched a specific COG, that is listed here. There is a website describing all of the COGS here.
7. `product`: The putative annotation for your ORF.

2.5 Annotate your own project datasets

Now that you have the general lay of the land, you're now going to annotate your own project dataset assemblies using `Prokka`.

2.5.1 30. Check on your project assemblies

Hopefully your project assemblies are done by now. Log back into your screen session (`screen -r assembly`). When IDBA-UD finishes, it will say:

```
invalid insert distance
Segmentation fault
```

This is because we gave IDBA-UD single-end reads instead of paired-end reads. Never fear! Despite appearances, your assembly actually did finish. However, the use of single-end reads instead of paired-end reads means that your assemblies will NOT have scaffolds, because we couldn't take advantage of the paired-end data to make scaffolds. So, your final assembly is just the final contig file from the longest kmer, called `contig-100.fa`.

2.5.2 31. Rename assembly

When your project assembly is done, change the name of your final assembly from `contig-100.fa` to a name that starts with your dataset number, followed by `_assembly.fasta`. For example, you might call it something like `ERR598966_assembly.fasta`.

This way we will have standardized names and save ourselves future heartache. While in your project_directory, type:

```
mv ERR598966_assembly/contig-100.fa ERR598966_assembly.fasta
```

(You learned how to do this already– see your Unix cheat sheet.)

2.5.3 32. Run anvi-script-reformat-fasta

Run `anvi-script-reformat-fasta` on your completed project assemblies (see step 19 above). For example:

```
anvi-script-reformat-fasta ERR598966_assembly.fasta -o ERR598966_assembly_reformatted.
↪fa -l 0 --simplify-names
```

2.5.4 33. Copy data to shared folder

Please copy your reformatted project assemblies and put them into the folder that we'll be sharing as a class, substituting the placeholder below with the name of your assembled file. This will be the shared folder for all of the data that you generate as a class. I made a folder called `assemblies` that you will put your assembled contigs into.

```
cp [name of your formatted, assembled file] /Accounts/Genomics_Bioinformatics_shared/
↪assemblies
```

2.5.5 34. Run Prokka on your project dataset

If you aren't in your project directory right now, move into it and start `prokka`. Remember that you're still in screen. This shouldn't take too long, but if you log out of screen, don't forget to re-enter and then terminate the screen session!

```
cd ~/project_directory
screen
prokka [your assembly reformatted] --outdir prokka_project
Ctrl+A d
```

2.5.6 35. Share your data

When Prokka is done, please copy your completed tsv and faa files to the shared class folder, and rename it with the name of your project dataset so it's recognizable. for example:

```
cp PROKKA_09252018.tsv /Accounts/Genomics_Bioinformatics_shared/PROKKA_results/
↪ERR598995_ORFs.tsv
cp PROKKA_09252018.faa /Accounts/Genomics_Bioinformatics_shared/PROKKA_results/
↪ERR598995_ORFs.faa
```

2.6 Playing with "big data"

Congratulations! You now have annotations for your entire project dataset. This means you have annotations for *thousands* of genes in your dataset. Feeling overwhelmed? Feeling excited about the endless possibilities? Feeling hungry for some questionable yet tasty LDC food? If it's one of the first two– welcome to big data! If it's the last one, maybe lab has gone on for too long.

We're almost done. But first we're going to see an example of how we might analyze this kind of data.

2.6.1 36. COG categories

First, there's a text file on the server that assigns every single COG in the COG database to a specific gene category, like `Translation, ribosomal structure, and biogenesis`. You can see it by typing this:

```
less /usr/local/data/Python_scripts/COG_numbers_categories_annotations.txt
```

2.6.2 37. Getting COG categories of your genes

Let's say you want to know how many of the genes in your dataset are responsible for translation, how many are for energy metabolism, how many are viruses, etc. Fortunately, there is a magic script available for you to do that on the server. Change directories into wherever your Prokka results are and type this:

```
get_ORF_COG_categories.py [name of your Prokka tsv file]
```

And voilà! You'll get a new file that ends in `_cog_categories.txt` with a list of all the different COG categories and the number of genes that fall into that category. Some of the COGs fall into multiple categories, denoted with, for example, `Signal_transduction_mechanisms/Transcription`. Imagine the possibilities! You can investigate so many questions about gene function in your dataset!

2.6.3 38. Share your data

Please copy your COG categories file into the shared class folder so others can access it, and while you're at it, change the name so it's easily recognizable. For example:

```
cp PROKKA09252019_cog_categories.tsv /Accounts/Genomics_Bioinformatics_shared/PROKKA_
↪results/ERR590988_cog_categories.txt
```

Obviously, please substitute the names of your own files for the ones listed above.

2.6.4 39. Exit

If you haven't killed your screen yet, you should do so. As you did above, while in your screen session, type: `Ctrl A` and then `k`. The the computer will say: `Really kill this window [y/n]`. You type: `y`.

When you are all done with your work on the server, you can log off the server by typing:

```
exit
```

2.7 Lab assignment this week

Write a mini research question about the COG categories you got from your ORF annotations. You can compare your sample to someone else's. For example, you might ask, "Is there a difference in the number of genes related to the mobilome/prophage in the surface ocean compared to the mesopelagic zone?" Make an Excel bar plot based on your COG category results that demonstrates the answer to your question. In a paragraph or so, explain your results and speculate as to why your results look they way they do. You should cite at least one paper from the literature that is relevant to your results. Submit this on Moodle by lab time next week.

I prefer to grade these blind, so please put your student ID number, rather than your name, on your assignment. (This applies to all future assignments as well.)

CHAPTER 3

Week 3: Local alignments and sequence search with BLAST, global alignments with MUSCLE, and making trees with RAxML

Rika Anderson, Carleton College

3.1 Logging in to the remote server

3.1.1 1. Login

If you're in a lab on campus, boot as a Mac on the lab computer.

3.1.2 2. Terminal

Find and open the Terminal application (`Applications/Utilities`). If you're on a PC, open your Ubuntu terminal or your PuTTY terminal.

3.1.3 3. Connect to baross

Log on to the server using your Carleton username and password.

```
ssh [username]@baross.its.carleton.edu
```

3.2 Using BLAST

Last week, we annotated thousands of ORFs in one go. As awesome as that was, you may have noticed that there were a lot of proteins labeled as "hypothetical." Let's see if we can learn more about some of those genes by using BLAST, which is a tool that every bioinformatician should have in their toolkit.

3.2.1 4. Copy sequence of your first hypothetical ORF

Will use BLAST to compare your sequences against the National Center for Biotechnology Information (NCBI) non-redundant database of all proteins. This is a repository where biologists are required to submit their sequence data when they publish a paper. It is very comprehensive and well-curated.

Change directory into your toy dataset Prokka results, and take a look at your Prokka amino acid sequence file again using the program `less`. (Again, substitute the actual names of your Prokka folder and files. Use tab complete as a shortcut!)

```
cd toy_dataset_directory/ORF_finding/prokka_toy
less PROKKA_09252018.faa
```

Copy the sequence of the **third** ORF in your file, the first one labeled as a 'hypothetical protein'. You can include the first line that includes the > symbol.

3.2.2 5. Navigate to NCBI site

Navigate your web browser to the BLAST suite home page. Select Protein BLAST (blastp).

3.2.3 6. Paste sequence

Copy your sequence in to the box at the top of the page, and then scroll to the bottom and click "BLAST." Give it a few minutes to think about it.

3.2.4 7. Run BLAST via the web browser

While that's running, open a new tab in your browser, navigate to the BLAST suite home page, and try blasting your protein using `tblastn` instead of `blastp`.

What's the difference?

`blastp` is a protein-protein blast. When you run it online like this, you are comparing your protein sequence against the National Centers for Biotechnology Information (NCBI) non-redundant protein database, which is a giant database of protein sequences that is "non-redundant"—that is, each protein should be represented only once.

In contrast, `tblastn` is a translated nucleotide blast. You are blasting your protein sequence against a translated nucleotide database. When you run it online like this, you are comparing your protein sequence against the NCBI non-redundant nucleotide database, which is a giant database of nucleotide sequences, which can include whole genomes.

Isn't this a BLAST?!? (Bioinformatics jokes! Not funny.)

3.2.5 8. Post-lab questions

For your post-lab assignment, answer the following questions in a Word document that you will upload to the Moodle:

Question 1: Check for understanding:

Take a look at your results from blastp and tblastn. In your own words, what is the difference between the two, and why do the results look different? Explain in what scenarios you might choose to use blastp over tblastn.

3.2.6 Doing a BLAST against your own dataset

We just learned how to compare a sequence of interest against all proteins in the NCBI database. But let's say you just isolated a protein sequence that you're interested in, and want to know if any of the ORFs or contigs in your dataset has a match to just that sequence. In that case, you would want to blast against your own sequence set, not against the giant NCBI database. Here's how you do that.

First, make sure you are in the correct directory.

```
cd ~/toy_dataset_directory/ORF_finding/prokka_toy
```

9. Make blast database

Turn your ORF dataset into a BLAST database. Here is what the flags mean:

- `makeblastdb` invokes the command to make a BLAST database from your data
- `-in` defines the file that you wish to turn into a BLAST database
- `-dbtype`: choices are "prot" for protein and "nucl" for nucleotide.

```
makeblastdb -in PROKKA_09252018.faa -dbtype prot
```

10. Find protein

Your ORFs are ready to be BLASTed. Now we need a protein of interest to search for. Let's say you are interested in whether there are any close matches to CRISPR proteins in this dataset. The Pfam database is a handy place to find 'seed' sequences that represent your protein of interest. Navigate your browser to this Pfam website.

This takes you to a page with information about the RAMP family of proteins, the "Repair Associated Mysterious Proteins." While indeed mysterious, they turn out to be CRISPR-related proteins.

11. Find protein (cont.)

Click on "2871 sequences" near the top. If the link doesn't work, click on "Alignments" on the side bar.

12. Generate file

Under "format an alignment," select your format as "FASTA" from the drop-down menu, and select gaps as "No gaps (unaligned)" from the drop-down menu. Click "Generate."

13. View file

Take a look at the Pfam file using less or using a text editing tool on the local computer. It is a FASTA file with a bunch of "seed" sequences that represent a specific protein family from different organisms. If you want to learn more about a specific sequence, you can take the first part of the fasta title (i.e. "Q2JRT5_SYNJA") and go to this website and paste it in the box. Then select from "UniProt KB AC/ID" to "UniProtKB" in the drop-down menu on the bottom (this should be the default). You will get a table with information about your protein and which organism it comes from (in this case, a type of bacterium called *Synechococcus*).

14. Transfer file

We need to put the file that you downloaded to your local computer onto the remote server. As a reminder, we do this using `scp` (secure copy). It lets you copy files to and from your local computer on the command line. You have to know the path to the file you want to copy on baross, and the path to where you want to put it on your local computer. It's a lot like cp, except it allows you to copy things between computers.

Open up a new Terminal window, but **DON'T** log in to baross on that one. Navigate to wherever your PFAM file is, then copy it onto the server. We're going to put the file in your ORF_finding directory:

```
cd ~/Downloads
scp PF03787_seed.txt [username]@baross.its.carleton.edu:~/toy_dataset_directory/ORF_
↪finding/prokka_toy
```

It will ask you for your password; type it in and hit "Enter."

15. BLAST it

Now, BLAST it! There are many parameters you can set. The following command illustrates some common parameters.

```
blastp -query PF03787_seed.txt -db PROKKA_09252018.faa -evalue 1e-05 -outfmt 6 -out␣
↪PF03787_vs_toy_ORFs.blastp
```

- `blastp` invokes the program within the blast suite that you want. (other choices are `blastn`, `blastx`, `tblastn`, `tblastx`.)
- `-query` defines your blast query– in this case, the Pfam seed sequences for the CRISPR RAMP proteins.
- `-db` defines your database– in this case, the toy assembly ORFs.
- `-evalue` defines your maximum e-value, in this case 1x10-5
- `-outfmt` defines the output format of your blast results. option 6 is common; you can check out this link for other options.
- `-out` defines the name of your output file. I like to title mine with the general format `query_vs_db.blastp` or something similar.

16. Examine results

Now let's check out your blast results. Take a look at your output file using `less`. (For easier visualization, you can also either copy your results (if it's a small file) and paste in Excel, or transfer the file using `scp` and open it in Excel.)

17. Column descriptions

Each blast hit is listed in a separate line. The columns are tabulated as follows, from left to right:

1. query sequence name
2. database sequence name
3. percent identity
4. alignment length
5. number of mismatches
6. number of gaps

7. query start coordinates
8. query end coordinates
9. subject start coordinates
10. subject end coordinates
11. e-value
12. bitscore

19. BLAST comparison

Let's try this another way. Run your blast again, but this time use a bigger e-value cutoff.

```
blastp -query PF03787_seed.txt -db PROKKA_09252018.faa -evalue 1e-02 -outfmt 6 -out␣
↪PF03787_vs_toy_ORFs_evalue1e02.blastp
```

Answer the following question on the Word document you will submit to Moodle:

Question 2: Check for understanding: In a few sentences/a short paragraph, describe: How do these BLAST results differ from your previous BLAST? What can you infer about what the e-value is based on these results? Explain in your own words why you might choose to use this method of BLAST (with a query sequence against your own dataset) rather than BLAST a query sequence against the nr database on the NCBI BLAST website.

20. Try another gene

Now let's try a different gene. Pfam is a good place to find general protein sequences that are grouped by sequence structure, with representatives from many different species. Let's say you're interested in finding a specific sequence instead.

1. Go to the NCBI Protein database.
2. Let's look for the DNA polymerase from *Thermus aquaticus*, the famous DNA polymerase that is used in PCR. Type "DNA polymerase Thermus aquaticus' in the search bar at the top.
3. Click on the first hit you get. You'll see lots of information related to that sequence.
4. Click on "FASTA" near the top. It will give you the FASTA sequence for that protein (protein rather than DNA).
5. Copy that sequence and paste it into a new file on baross. Here is one way to do that: use `nano` to create a new file (see command). Once in `nano`, paste your sequence into the file. Type `Ctrl+O`, `Enter`, and then `Ctrl+X`.

```
nano DNA_pol_T_aquaticus.faa
```

21. BLAST

Now you have a sequence file called `DNA_pol_T_aquaticus.faa`. BLAST it against your toy dataset:

```
blastp -query DNA_pol_T_aquaticus.faa -db PROKKA_09252018.faa -outfmt 6 -out DNA_pol_
↪T_aq_vs_toy_ORFs.blastp
```

22. Examine results

Take a look at your blastp output using less. Notice the e-value column, second from right. Most of your e-values for this range between 2.3 (at best) and 7.4 (at worst). These e-values are TERRIBLE. This means you probably didn't have very good hits to this protein in your dataset. (What constitutes a "good" e-value is debatable and depends on your application, but a general rule of thumb is for it to be 1e-05 or lower.)

23. Post-lab questions

Now choose your own protein and BLAST it against your dataset. **Answer the following questions on the Moodle Word document:**

Question 3: Check for understanding:

In a few sentences/a short paragraph, describe the protein you chose and how you found the sequence for that protein. Show the command you executed for the blast, and show the first three matches (if there were any) with an e-vaue cutoff above 1e-05.

3.3 Making An Alignment

OK, so we've learned how to do sequence search using BLAST, which is a *local* alignment search tool. Now we're going to learn how to make global alignments using MUSCLE, and then use those *global* alignments to make phylogenetic trees in order to learn about how genes are related.

We'll start by creating a multiple sequence alignment with MUSCLE, and then we will make bootstrapped maximum likelihood phylogenetic trees with RAxML. We'll use the Newick files generated by RAxML to visualize trees in an online tree visualization tool called the Interactive Tree of Life (iToL). We'll do this using toy datasets, and then try out some trees on your project datasets.

3.3.1 24. Aligning sequences

First we have to make a multiple sequence alignment with the sequences we wish to make into a tree. This could include any gene of interest. We're going to start by aligning a toy dataset made of genes that are part of photosystem II in photosynthesis. This file contains many sequences of the same photosystem II protein from different species.

Make a new directory within your toy dataset directory for making alignments and trees, then copy the toy dataset from the data directory to your toy dataset directory.

```
mkdir toy_dataset_directory/alignments_and_trees
cd toy_dataset_directory/alignments_and_trees
cp /usr/local/data/toy_datasets/toy_dataset_PSII_protein.faa .
```

3.3.2 25. Align with MUSCLE

Now, we make a multiple sequence alignment using `muscle`. `muscle` uses dynamic programming to make global alignments of multiple sequences, like we did in class. It's remarkably easy and fast.

```
muscle -in toy_dataset_PSII_protein.faa -out toy_dataset_PSII_protein_aligned.afa
```

What this means:

`muscle` is the name of the alignment program

`-in` defines the name of your input file, which can be either DNA or protein

`-out` defines the name of your output file. I like to give them an easy-to-recognize name with the extension `.afa`, which stands for "aligned fasta file."

3.3.3 26. Secure copy to a local computer

Let's take a look at your alignment. I like to use the program Seaview to do this, and Seaview should be on your local computer (if not, go back to the Week 1 pre-lab for instructions). You will have to copy your file to your local computer using `scp`. (As before, substitute `[username]` with your own username.)

```
scp [username]@baoss.its.carleton.edu:/Accounts/[username]/toy_dataset_directory/
↪alignments_and_trees/toy_dataset_PSII_protein_aligned.afa ~/Desktop
```

This lets you securely copy the aligned protein file from baross to your local computer. Substitute [username] with your own username.

If you wanted to securely copy any file from your local computer to baross, use the command below. It's easy to find the path of where you want to put things– simply navigate to where you want to put it on the server, and then either copy everything before the $ or just type `pwd`.

```
scp ~/Desktop/[some_file.txt] [username]@baross.its.carleton.edu:[path_of_your_
↪destination_directory]
```

3.3.4 27. Visualize with Seaview

Open the application called "Seaview" and drag your file over to the alignment window. You should see something like this.

seaview screenshot

Seaview shows the names of the sequences to the left. The letters to the right are the amino acids in your sequence, color-coded to make the alignment easier to see. You can easily see that some regions of the sequence are more highly conserved than others, and that some species appear to have an insertion or deletion in specific regions of the sequence. Note that this is an amino acid alignment, not a nucleotide alignment. (You could easily use MUSCLE to align nucleotides as well.)

3.4 Making a Tree

Now we're going to turn this alignment into a phylogenetic tree. We're going to use a software package called RAxML, which is a commonly used tree-building software package that uses the maximum likelihood method to build trees.

3.4.1 28. Formatting

First, we have to convert the aligned fasta file into a format that the tree-building software (RAxML) can handle. (A lot of the work of a bioinformatician is converting files from one format into the other.) I wrote a small Python script to convert your aligned fasta (.afa) file into Phylip file (.phy), which is what RAxML requires for input. Invoke it with the name of the script, followed by the aligned fasta file that you'd like to convert, like this:

```
convert_afa_to_phy.py toy_dataset_PSII_protein_aligned.afa
```

3.4.2 29. Looking at your Phylip file

The script should have outputted a file that ends in .phy. Take a look at it to see what the format looks like.

The top of a Phylip file indicates the number of sequences (in this case, 17) and the number of base pairs in the aligned sequences (571). It's followed by the first 54 bases of each sequence, broken up by a space every 10 letters. Every new line shows a new sequence (from a different species or organism). The next 54 letters of each sequence starts after the first set, and continues like this for the rest of the file. (Yes, it's weird, but a lot of tree-building programs use this format.)

```
less toy_dataset_PSII_protein_aligned.afa.phy
```

3.4.3 30. Build your tree!

Now let's make a tree. Type this:

```
raxmlHPC-PTHREADS-AVX -f a -# 20 -m PROTGAMMAAUTO -p 12345 -x 12345 -s toy_dataset_
↪PSII_protein_aligned.afa.phy -n toy_dataset_PSII_protein.tree -T 4
```

What this means:

`-raxmlHPC-PTHREADS-AVX` is the name of the software package. This one is configured for the processors that are specific to this server.

`-f a` allows for rapid bootstrapping. This means RAxML will do a maximum likelihood search based on your protein sequences, and then bootstrap it as many times as you wish.

`-# 20` tells the program to bootstrap 20 times.

`-m PROTGAMMAAUTO` tells the program that these are protein sequences, and tells the program how to model protein evolution. To get into this is beyond the scope of this class, but fortunately RAxML is able to automatically choose the best one for us based on the number of sequences and the type of data we have.

`-p` and `-x` provide seed numbers so that the program can generate random numbers for the bootstrapping process.

`-s` gives the name of your input Phylip file

`-n` gives the name of your output Newick file, which will be made into a tree.

`-T` determines the number of threads. This is sort of like determining how many processors you'll use up for this process. Today, we'll use 4. Please don't use more than this without asking first.

NOTE: "bootstrapping" is a statistical test used to assess the reliability of your tree topology (its shape). We'll talk about this more in class next week.

31. Tree output

You've made a tree! Congratulations! Let's look at the raw RAxML output. You should have some files called:

`RAxML_bestTree.toy_dataset_PSII_protein.tree` `RAxML_bipartitionsBranchLabels.toy_dataset_PSII_protein.tree` `RAxML_bipartitions.toy_dataset_PSII_protein.tree` <– this is the one you want, because it gives you bootstrap values at the nods on the tree. (more on bootstrapping next week.) `RAxML_bootstrap.toy_dataset_PSII_protein.tree` `RAxML_info.toy_dataset_PSII_protein.tree`

Take a look at the bipartitions file using `less`.

This is a Newick file, and it's a common format for phylogenetic trees.

32. Visualizing the tree

Copy your Newick file (toy_dataset_PSII_protein.tree) to your local computer using `scp`. Open up a web browser on your local computer and navigate to the ITOL website and create an account for yourself.

Click on "Upload tree files" and find your tree file and upload it.

Click on your tree. You should see it open in your window. You can play around with the settings on the right—you can make it circular or normal, you can display the bootstrap values as symbols or as text, and if you click on the leaves (the tips) or the nodes (the interior bifurcations) of the tree, you can color code them. Play around with it, and then click on the "Export" tab, choose the PDF format, and click "Export." It should pop up in a new tab. Download it and include it in your postlab writeup this week as "Figure 1." Be sure to include a figure caption.

3.5 Asking Biological Questions with Trees

Now that you know how to make an alignment, create a phylogenetic tree, and visualize it, we're going to ask a biological question that can be answered with trees:

What evolutionary clade do photosynthesis genes from surface waters fall into?

We would probably expect to see more photosynthesis genes related to photosynthetic bacteria, and perhaps seaweed, in the surface oceans, but not photosynthesis genes related to terrestrial plants. We can make a phylogenetic tree to determine whether that's true, and which specific organisms they're most closely related to.

3.5.1 33. Exploring the KEGG database

First, we need to identify a target protein within the photosynthetic apparatus. For this, let's go to the KEGG Pathways website, which has a description of major metabolic pathways and the proteins that comprise them. It's a lot like the COG database, but better updated, and with a lot more information.

Link to KEGG database

Click on 'Energy' under 'Metabolism' and then click on 'Photosynthesis.' You should see a depiction of the photosynthetic proteins. Let's choose a protein within the photosynthetic apparatus that might be of interest. For the sake of this example, let's choose psbA: it's part of the photosystem II p680 reaction center D protein. Click on the box labeled 'PsbA' beneath the figure. Now you'll see lots of information about that particular gene. You can use the KEGG website like this to figure out which proteins or genes might be of interest for your projects later on.

3.5.2 34. Getting some sequences from Pfam

However, it will be more useful for us to have a nice set of seed sequences to align against– so let's go back to Pfam. Click on this link.

Click on '1953' sequences near the top click on 'FASTA' from the dropdown menu, click 'No gaps (unaligned)' from the drop-down menu, and save the resulting file on baross within a new directory in your toy dataset directory. It would probably be best to create a separate folder for this. Use scp to transfer it over to that new directory, or you could copy the file and create a new file on baross using nano and copy it there (many options!).

```
cd
mkdir toy_dataset_directory/tree_exercise_1
cd toy_datset_directory/tree_exercise_1
```

3.5.3 35. Copy Tara data

First we have to find matches to that gene in our dataset. So, we have to BLAST this dataset against the dataset of interest. We're going to use ORFs from a Tara Oceans dataset from the North Atlantic surface oceans off the East Coast. Copy it over to your current directory.

```
cp /usr/local/data/toy_datasets/ERR598983_ORFs.faa .
```

3.5.4 36. Make BLAST database

Make this dataset into a BLAST database.

```
makeblastdb -in ERR598983_ORFs.faa -dbtype prot
```

3.5.5 37. BLAST

BLAST it.

```
blastp -query PF00124_seed.txt -db ERR598983_ORFs.faa -outfmt 6 -evalue 1e-05 -out␣
↪PF00124_vs_ERR598983_ORFs.blastp
```

3.5.6 38. Look at BLAST results

Take a look at the results. You'll notice lots of hits. However, you'll see that all of the hits were to a single ORF: contig-100_83_1.

```
less PF00124_vs_ERR598983_ORFs.blastp
```

3.5.7 39. Search and copy from terminal using less

Let's extract it. Open up the original file with the open reading frames, and copy the sequence for contig-100_83_1.

```
less ERR598983_ORFs.faa
```

You can search for a specific contig by typing: `/contig-100_83_1`

3.5.8 40. Add to FASTA file

Now let's add it to the original Pfam file and make a tree from all of those sequences. First, make a copy of the original Pfam file and give it a new name. Then use nano to open it up and paste the ORF sequence to the bottom.

```
cp PF00124_seed.txt PF00124_seed_plus_new_ORF.fasta
nano PF00124_seed_plus_new_ORF.fasta
```

3.5.9 41. Align with MUSCLE

Make a multiple sequence alignment with `muscle`.

```
muscle -in PF00124_seed_plus_new_ORF.fasta -out PF00124_seed_plus_new_ORF_aligned.afa
```

3.5.10 42. Convert to Phylip

Convert your aligned FASTA file to a Phylip file.

```
convert_afa_to_phy.py PF00124_seed_plus_new_ORF_aligned.afa
```

3.5.11 43. Make tree

Make your tree. When it's done, open it up in IToL and make a pretty tree image. (The RAxML run might take a few minutes, so you might get started on the last question for the day while this runs.)

```
raxmlHPC-PTHREADS-AVX -f a -# 20 -m PROTGAMMAAUTO -p 12345 -x 12345 -s PF00124_seed_
↪plus_new_ORF_aligned.phy -n PF00124_seed_plus_new_ORF.tree -T 4
```

3.5.12 44. Visualize with ITOL

Visualize your tree with ITOL.

3.5.13 45. Postlab questions: Check for understanding

Include an image of your tree and call it 'Figure 2.' Include a figure caption.

3.6 Post-lab assignment

For your post-lab assignment, compile all of the Moodle questions above into a single document. There are three 'check for understanding' questions listed above, and two figures. In addition, answer the critical thinking question below.

Critical thinking question:

Develop a question about your project dataset that you can answer using BLAST, making trees, or both. For example, your question could be something like: 'how many ORFs in my dataset have a match to a specific type of photosynthesis gene?' or 'I found a cytochrome c gene in my dataset. Where does it fit on a tree of other cytochrome C genes from Pfam?'

For this week's postlab assignment, describe:

-What question did you ask?

-How did you go about answering it? (Write this like you would a mini Materials and Methods section: include which databases you searched, which software packages you used, and which important flags you used in your commands. You should include enough information for an intelligent researcher to be able to replicate your results.)

-What were your results? Describe them. (Write this like you would a mini Results section. Include a figure if appropriate.)

-What does this tell you about your project dataset? (Think of this as a mini Discussion section: I'm looking for evidence that you thought about your results and how they connect more broadly to some ecological or evolutionary pattern in your dataset.)

Compile all of this together into a Word (or similar) document and submit via Moodle by lab next week. **I prefer to grade these blind, so please put your student ID number, rather than your name, on your assignment. (This applies to all future assignments as well.)**

CHAPTER 4

Week 4: Mapping with bowtie2

4.1 1. Log in to the remote server

Open up your Terminal window and ssh in to baross.

4.2 Mapping with a toy dataset

4.2.1 2. Make new directory

We're going to start by mapping the sequencing reads from a genome sequence of a type of archaeon (*Sulfolobus acidocaldarius*– you've seen it before) against a scaffold from a very closely related species.

The sequencing reads from from one strain of *Sulfolobus acidocaldarius*, and the reference sequence that they are mapping to is from a very closely related strain of *Sulfolobus acidocaldarius*.

In your home directory, make a new directory called "mapping," then change into that directory.

```
mkdir mapping
cd mapping
```

4.2.2 3. Copy data

Copy this week's toy datasets into your directory. You're going to copy: -the reference sequence (toy_dataset_contig_for_mapping.fasta) -the reads that you will map to the reference (toy_dataset_reads_for_mapping.fasta)

```
cp /usr/local/data/toy_datasets/toy_dataset_reads_for_mapping.fasta .
cp /usr/local/data/toy_datasets/toy_dataset_contig_for_mapping.fasta .
```

4.2.3 4. Build index

You're going to be mapping short reads to a longer reference sequence. The first thing you have to do is prepare an index of your reference so that the mapping software can map to it.

```
bowtie2-build toy_dataset_contig_for_mapping.fasta toy_dataset_contig_for_mapping.
↪btindex
```

-`bowtie2-build` is the program that indexes your reference. -The first argument gives the reference dataset name. -The second argument provides the name you want to give to the index.

4.2.4 5. Map!

Now, map! This will take a few steps.

First, you make what is called a SAM file. It's a human-readable version of a BAM file, which we learned about previously in class.

-`bowtie2` is the name of the mapping program. `--x` is the flag that provides the name of the index you just made. `--f` means that the reads you are mapping are in fasta, not fastq, format. `--U` means that the reads are not paired. (They aren't in this dataset.) `--S` provides the name of your output file, which is in SAM format.

```
bowtie2 -x toy_dataset_contig_for_mapping.btindex -f -U toy_dataset_reads_for_mapping.
↪fasta -S toy_dataset_mapped_species1.sam
```

4.2.5 6. Look at SAM file output

If you look at the output file with `less`, you can see that it is human-readable (sort of). It tells you exactly which reads mapped, and where they mapped on the reference, and what the differences were between the reference and the mapped reads. This can sometimes be useful if you want to parse it yourself with your own scripts– but there's a whole suite of tools in a package called `samtools` that we'll rely on to do that next.

4.2.6 7. samtools

Now you will use a package called `samtools` to convert the SAM file into a non-human-readable BAM file. You've heard of BAM files before– now you get to make one.

```
samtools view -bS toy_dataset_mapped_species1.sam > toy_dataset_mapped_species1.bam
```

-`samtools` is a package used to manipulate and work with mapping files. samtools view is one program within the whole samtools package. -The flag `-bS` is not BS! It tells samtools to convert a bam file to a sam file. (Bioinformatics jokes = still not very funny.)

4.2.7 8. samtools sort

And because this is so fun, we get to do some more bookkeeping. Sort your bam file so that later programs have an easier time parsing it:

```
samtools sort toy_dataset_mapped_species1.bam -o toy_dataset_mapped_species1_sorted.
↪bam
```

-`samtools sort` is the name of the program used for sorting -The first argument provides the name of the bam file you want to sort -The `-o` flag gives the name of the output file you want.

4.2.8 9. Index the reference with samtools

In order to visualize your mapping, you have to **index your reference**. Because indexing is SO MUCH FUN. This time we are indexing with samtools instead of bowtie2.

(**NOTE!!**: you only need to do this step if you're going to visualize your mapping with IGV, as we're about to do now. In the future, if you don't intend to visualize your mapping, then you don't need to bother with this step.)

```
samtools faidx toy_dataset_contig_for_mapping.fasta
```

-`samtools faidx` is the name of the program that indexes the reference. -The first argument provides the name of the index, which should be your reference file.

4.2.9 10. Index the bam file

Almost there! Now you index the bam file that you just made because WE LOVE INDEXING.

```
samtools index toy_dataset_mapped_species1_sorted.bam
```

-`samtools index` is the name of the program that indexes the bam files. -The first argument provides the name of a sorted bam file.

4.2.10 11. Copy to local computer

Now we're going to visualize this. Copy the entire "mapping" folder over to your local computer using scp.

```
scp -r [your username]@baross.its.carleton.edu:/Accounts/[your username]/toy_dataset_
↪directory/mapping/ ~/Desktop
```

Remember, to use scp, you should open a new Terminal window that is NOT logged in to baross. The above command copies the folder at toy_dataset_directory/mapping to your local computer's Desktop.

4.2.11 12. Visualize in IGV

Find the IGV Viewer on your local computer and open it. **First,** click 'Genomes' –> 'Load Genome from File' and find your reference file ('toy_dataset_contig_for_mapping.fasta'). **Next,** click 'File' –> 'Load from File' and open your sorted bam file. You should be able to visualize the mapping. Along the top, you'll see the coordinates of your reference sequence. Below that, you'll see a graph showing the coverage of each base pair along your reference sequence. Below that, you'll see each read mapped to each position. The arrows indicate the direction of the read; white reads are reads that mapped to two different locations in your reference. Single nucleotide variants in the reads are marked with colored letters; insertions are marked with a purple bracket, and deletions are marked with a horizontal black line. More information can be found at the link below.

Link to IGV viewer

4.2.12 13. Compare mappings

We're going to compare and contrast this mapping with another one. Now we're use the sequencing reads from a third very closely related strain of *Sulfolobus acidocaldarius*, and we're going to map those reads to the original reference sequence so that we can compare the mapping.

All of the commands are listed below. First, you will copy the second file to your directory (see below). You have already indexed the reference file. Then you map, convert SAM to BAM, sort, and then index.

```
cp /usr/local/data/toy_datasets/toy_dataset_reads_for_mapping_species2.fasta .
bowtie2 -x toy_dataset_contig_for_mapping.btindex -f -U toy_dataset_reads_for_mapping_
↪species2.fasta -S toy_dataset_mapped_species2.sam
samtools view -bS toy_dataset_mapped_species2.sam > toy_dataset_mapped_species2.bam
samtools sort toy_dataset_mapped_species2.bam -o toy_dataset_mapped_species2_sorted.
↪bam
samtools index toy_dataset_mapped_species2_sorted.bam
```

4.2.13 14. Visualize new mapping

Copy the new data files to your local computer using scp like in step 11, and then visualize both of them in IGV viewer. Since you have already loaded the reference file and reads from your first mapping, all you have to do is click 'File' –> 'Load from File' and click on your new bam file. You should be able to see them side by side.

Answer these questions for this week's postlab assignment.

Check for understanding, Question 1: Describe the large-scale differences between the mapped reads from species 1 and species 2, and explain what this mapping tells us about the relative genome structure of the two genomes that we mapped. If we compared this genomic region in a dot plot, what would it look like?

4.3 Mapping your project datasets

Now we're going to map your project datasets. Remember that these are metagenomes, not a genome, so the data will be a bit more complex to interpret.

We're going to map your raw reads against your assembled contigs. Why would we do this, you ask? A few reasons:

-to look for single nucleotide variants in specific genes.

-to quantify the relative abundances of different genes, and determine whether specific genes have better coverage than others.

-to quantify the relative abundances of specific taxa, and determine whether specific taxa are more abundant than others.

As you consider this, answer the following question for this week's lab questions:

Check for understanding, Question 2: If you wanted to quantify the relative abundances of specific genes in your sample, why couldn't you simply count the number of times your gene appears in your assembly?

4.3.1 15. Map to project datasets

Change directory into your project dataset directory folder. We're going to map your raw reads against your assembled contigs (not your ORFs). Make sure you know where your project assembly is and where your raw reads are. Follow the instructions to map your raw reads back to your assembled contigs. For example, if you were mapping the dataset ERR599166_1mill_sample.fasta and your assembly was called ERR599166_assembly_formatted.fa, you might do something like this (below). Please be sure to use the assembled files that you've already run through anvi-script-reformat-fasta, which you should have done in our first computer lab.

One more thing. Your project datasets are very large– you each have 10 million reads. So mapping will take longer than for the toy datasets. So we're going to add an extra flag (-p) to the mapping step to tell the computer to use more than one processor so the process goes faster. You'll each use 5 for this process. The mapping may still take about 5-10 minutes, so have patience!

An example set of commands is shown below. **Remember to replace the dataset names here with your own project datasets!**

```
bowtie2-build ERR599166_assembly_formatted.fa ERR599166_assembly_formatted.btindex
bowtie2 -x ERR599166_assembly_formatted.btindex -f -U ERR599166_sample.fasta -S␣
↪ERR599166_mapped.sam -p 5
samtools view -bS ERR599166_mapped.sam > ERR599166_mapped.bam
samtools sort ERR599166_mapped.bam -o ERR599166_mapped_sorted.bam
samtools faidx ERR599166_assembled.fa
samtools index ERR599166_mapped_sorted.bam
```

Note that this command assumes that your raw reads and your assembly are in the same directory you're in. If they are not, you will need to either copy them over or use the correct path in your commands. (A reminder: the path simply gives directions to the computer for where to find a file.) For example, if you are in a mapping directory, your reads file is one directory up in the file hierarchy, and your assembled reads are in your assembly file, you might have to type something like this:

```
bowtie2 -x ../assembly/ERR599166_assembled.btindex -f -U ../
ERR599166_1mill_sample.fasta -S ERR599166_mapped.sam
```

4.3.2 16. Visualize

When you visualize this in IGV, remember that you have multiple contigs. So you have to click the drop-down menu at the top and choose which contig you wish to visualize.

4.3.3 17. Calculating coverage- generate bed file

You were able to visualize the mappings in IGV, but sometimes you just want to have a number: for example, you might want to know the average coverage across a specific gene, and compare that to the average coverage of another gene in order to compare their relative abundances in the sample. So, next we're going to calculate gene coverages based on your mapping.

First we're going to make what's called a bed file. We will use it to find the average coverage of every single open reading frame in your dataset. Please make sure that your contigs have names that are something like c_00000000001 and your ORFs have names that are something like c_00000000001_1.

```
make_bed_file_from_gff_file_prokka.py [your gff file from prokka]
```

For example:

```
make_bed_file_from_gff_file_prokka.py PROKKA_09252018.gff
```

This will create a bed file that ends in .bed. You can take a look at it if you wish– it should have the contig name, the coordinates of your ORF, and the name of your ORF.

4.3.4 18. Run script to calculate coverage

Now run a script that will use your bed file and will calculate the read depth for every single ORF in your ORF file.

```
samtools bedcov [your bed file] [your sorted bam file] > [ an output file that ends␣
↪in _ORF_coverage.txt]
```

For example:

```
samtools bedcov ERR599166_assembled.bed ERR599166_mapped_sorted.bam > ERR599166_ORF_
↪coverage.txt
```

It is extremely important that you ran **both** bowtie2 and prokka on the reformatted assembly! If you get errors, this is probably because you did not run them against the reformatted assembly!!

How do you check that you ran it on the reformatted dataset? First, check your Prokka output:

```
less PROKKA_09252018.gff
```

It should look something like this, with the second column showing numbers like c_000000000001 and so on.

```
##gff-version 3
##sequence-region c_000000000001 1 2569
##sequence-region c_000000000002 1 1660
##sequence-region c_000000000003 1 1376
##sequence-region c_000000000004 1 1037
##sequence-region c_000000000005 1 945
##sequence-region c_000000000006 1 917
##sequence-region c_000000000007 1 902
##sequence-region c_000000000008 1 892
...
```

If it looks more like this, you did not run prokka against the reformatted dataset.

```
##gff-version 3
##sequence-region contig-100_0 1 44454
##sequence-region contig-100_1 1 44231
##sequence-region contig-100_2 1 35217
##sequence-region contig-100_3 1 29595
##sequence-region contig-100_4 1 29534
...
```

If it looks like this, then you should run `anvi-script-reformat` again and then run prokka again. For example:

```
anvi-script-reformat-fasta contig-100.fa -o ERR599899_assembled_reformatted.fa -l 0 --
↪simplify-names
prokka ERR599899_assembled_reformatted.fa --outdir prokka_ERR599899
```

You should also make sure that you ran bowtie2 against the reformatted version. How do you know? Go back up up to Step 13 in this week's protocol and check the assembly file with `less`. Make sure the contigs start with c_00000001 and not contig-100_0.

4.3.5 19. Calculate coverage in Excel

It will give you a file that ends in `ORF_coverage.txt`. Use scp to move it to your local computer, then open with Excel. It should give the name of your contig, the start coordinate, the stop coordinate, the name of your open reading frame, and then the sum of the per-base coverage. To get the average coverage, divide the sum of the per-base coverage by the difference between the stop and start coordinates. In other words, type this in the top column and then fill down to the bottom: =E1/(C1-B1)

Now you have the average coverage of every ORF for this particular bam file, and you should be able to match the name of your ORF from Prokka to the ORF in this output file.

4.3.6 20. Check for understanding

This is a really common type of analysis for 'omics-based studies– you can compare the coverage of specific genes of interest. For example, you might compare the coverage of genes related to photosythesis, respiration, and nitrogen fixation if you're interested in how abundant those metabolisms are in the community. We also use a very similar technique when we're doing expression analyses using something like RNASeq (more on that in coming weeks).

Check for understanding, Question 3:

As you scroll through the data file reporting the average coverage of all of your ORFs, which ORF had the highest coverage? What did it encode, according to your Prokka file or BLAST? In one or two sentences, speculate on why that gene may have had the highest coverage of all the genes in your dataset. NOTE! The genes with the highest coverage were probably really short and resulted from Illumina sequencing error– i.e., ATATATATATATAT. IDBA-UD orders contigs by length, so I recommend skipping the contigs that are really short (high numbers in the contig name) and find the contig with the highest coverage that was unlikely to be a sequencing error.

4.3.7 21. Copy your mapping files to the shared class folder

Please copy your bam files, bai files, bed files, and your ORF coverage files over to the class shared folder. Remember to give them names that are uniform and recognizable (see below example, and substitute your file names for the ones below).

```
cp ERR598995_mapped_sorted.bam /Accounts/Genomics_Bioinformatics_shared/mapping
cp ERR598995_mapped_sorted.bam.bai /Accounts/Genomics_Bioinformatics_shared/mapping
cp ERR598995_assembly_ORFs.bed /Accounts/Genomics_Bioinformatics_shared/mapping
cp ERR598995_ORF_coverage.txt  /Accounts/Genomics_Bioinformatics_shared/mapping
```

4.3.8 22. This week's postlab writeup

For this week's post-lab writeup:

Mini Research Question

Write either a question or generate a hypothesis about the relative coverage of this set of genes with respect to your project datasets. This question/hypothesis should include a comparison between your own project dataset and another dataset, and it should be couched within the larger ecological context.

You do have some information to act as a basis for your question or hypothesis. For example, the KEGG metabolisms page you saw last week provides good information about which genes are used in specific pathways. You also have some metadata related to your project sample in the Project Dataset Excel spreadsheet on the Moodle.

Example #1: I hypothesize that there will be lower coverage of genes related to photosynthesis (i.e. the psb genes) in the mesopelagic zone relative to the surface. This is because at the surface there will be more organisms that photosynthesize compared to the mesopelagic zone, where less light is available. Therefore, a lower proportion of genes in the microbial community in the mesopelagic zone will be related to photosynthesis compared to the surface, and therefore, fewer reads will map to photosynthesis genes in the mesopelagic zone.

Example #2: I hypothesize that there will be higher coverage of genes related to viruses in my sample relative to the deeper samples because there are more viruses in surface waters than in deeper waters, simply because there are more organisms to infect in surface waters.

Once you've identified the set of genes related to a specific metabolism/function/type of organism, and you have written a question or generated a hypothesis, find the average coverage to each of those ORFs in your dataset. Remember that more than one ORF may have that function.

Many of you will probably want to compare your mapping of your own reads to your own dataset to a mapping made by one of your classmates to their own dataset. The bam files and ORF coverage files should be saved in /Accounts/Genomics_Bioinformatics_shared/mapping.

Describe your results and create at least one graph to visualize those results. This should represent a mini 'Results' section in a lab report or paper. Interpret your results within the context of the ecosystem you are investigating. This should represent a mini 'Discussion' section in a lab report or paper.

Compile your 3 "check for understanding" questions and your mini research question together and submit on Moodle by lab time next week.

CHAPTER 5

Week 5: Binning genomes with anvi'o

5.1 Intro (Rika will go over this at the beginning of lab)

This week in lab we'll learn how disentangle individual microbial genomes from your mess of metagenomic contigs. We aren't going to use a toy dataset this week—-we're going straight into analysis with your metagenome datasets for your projects.

Genomes are disentangled from metagenomes by clustering reads together according to two properties: **coverage** and **tetranucleotide** frequency. Basically, if contigs have similar coverage patterns between datasets, they are clustered together; and if contigs have similar kmers, they will cluster together. When we cluster contigs together like this, we get a collection of contigs that are thought to represent a reconstruction of a genome from your metagenomic sample. We call these 'genome bins,' or 'metagenome-asembled genomes' (MAGs).

There is a lot of discussion in the field about which software packages are the best for making these genome bins. And of course, the one you choose will depend a lot on your dataset, what you're trying to accomplish, and personal preference. I chose anvi'o because it is a nice visualization tool that builds in many handy features.

I am drawing a lot of information for this tutorial from the anvi'o website. If you'd like to learn more, see this link.

5.2 Preparing your contigs database for anvi'o

5.2.1 1. ssh tunnel

Boot your computer as a Mac and use the Terminal to ssh in to baross. Anvio requires visualization through the server, so this week we have to create what is called an "ssh tunnel" to log into the server in a specific way. Substitute "username" below with your own Carleton login name.

NOTE: Each of you will be assigned a different port number (i.e. 8080, 8081, 8082, etc.). Substitute that number for the one shown below.

```
ssh -L 8080:localhost:8080 username@baross.its.carleton.edu
```

5.2.2 2. Make new folder

Make a new directory called "anvio" inside your project folder, then change into that directory.

```
cd project_directory
mkdir anvio
cd anvio
```

5.2.3 3. Copying data

Last week, we learned how to make BAM files. This week, we'll need them to make our bins, because they give us coverage information. You need BAM files that show the coverage of your own reads mapped against your own contigs, as well as other BAM files showing other people's reads mapped against your contigs.

Fortunately for you, I made all of these BAM files for you ahead of time. They are in this folder:

/workspace/data/Genomics_Bioinformatics_shared/Tara_mappings

Choose **three** mapping files from that folder. The ones you choose should all be from your own dataset. The names are like this, as an example:

`ERR599021_assembled_vs_ERR599013_reads_sorted.bam`

Where ERR599021 is the reference, and reads from ERR599013 were mapped to it. You want only the bam files with your **own** sample number as the **reference**.

You should also include the bam file of your own reads mapped against your own contig.

```
cp [path to sorted .bam files] .
cp [path to sorted .bai files] .
```

For example:

```
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599021_reads_sorted.bam
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599021_reads_sorted.bam.bai
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599013_reads_sorted.bam
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599013_reads_sorted.bam.bai
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599075_reads_sorted.bam
cp /workspace/data/Genomics_Bioinformatics_shared/Tara_mappings/ERR599021_assembled_
↪vs_ERR599075_reads_sorted.bam.bai
```

Finally, copy your contigs over as well. They are probably in your assembly directory. For example:

```
cp ../ERR598983_assembly/ERR598983_assembled_reformatted.fasta .
```

5.2.4 4. Get gene calls and annotations from Prokka

The first thing you have to do is make contigs database, which contains the sequences of your contigs, plus lots of information about those contigs. This includes information from Prokka– so we have to take the information from Prokka and put it in our contigs database.

First, you have to run a script on your Prokka files to convert them into a text file that we can import into anvi'o. Navigate to wherever your Prokka results are for your project assembly, and run a script to extract information from that file. Then copy it to your anvi'o folder and change directory to your anvio folder. For example:

```
cd prokka_project
gff_parser.py PROKKA_10072018.gff --gene-calls prokka-gene-calls.txt --annotation␣
↪prokka-gene-annot.txt
cp gene_calls.txt ../../anvio
cp gene_annot.txt ../../anvio
cd ../../anvio
```

5.2.5 5. Make the contigs database

Now, you make the contigs database. It will have lots of information about... well... your contigs.

`-anvi-gen-contigs-database` is the anvi'o script that makes the contigs database.

`--f` is the fasta file with your contigs that you have already assembled and fixed.

`--o` provides the name of your new contigs database.

`-external_gene_calls` provides the name of the Prokka file you just made so you can import the Prokka calls into your contigs database

`--ignore-internal-stop-codons` will ignore any internal stop codons in your gene calls. Sometimes these will get included in your Prokka results by accident, but for our purposes we can ignore them.

```
anvi-gen-contigs-database -f [your formatted, assembled contigs] -o contigs.db --
↪external-gene-calls prokka-gene-calls.txt --ignore-internal-stop-codons
```

5.2.6 6. Import the Prokka annotations

Import your prokka results like this:

```
anvi-import-functions -c contigs.db -i prokka-gene-annot.txt
```

5.2.7 7. Search for single copy universal genes

Now we will search our contigs for archaeal and bacterial single-copy core genes. This will be useful later on because when we try to disentangle genomes from this metagenome, these single-copy core genes can be good markers for how complete your genome is.

This process is slow, so we're going to run it on 5 CPUs rather than just 1. You can run it on screen in the background while you move forward with step 6. It should take a little under 10 minutes.

```
screen
anvi-run-hmms -c contigs.db -T 5
```

5.2.8 8. Determine taxonomy using Centrifuge

Now we are going to figure out the taxonomy of our contigs using a program called centrifuge. Centrifuge is a program that compares your contigs to a sequence database in order to assign taxonomy to different sequences within your metagenome. We're going to use it first to classify your contigs.

If you would like to know more, you can visit the anvi'o tutorial and the Centrifuge website.

First, export your genes from anvi'o.

```
anvi-get-sequences-for-gene-calls -c contigs.db -o anvio-gene-calls.fa
```

5.2.9 9. Run Centrifuge

```
centrifuge -f -x /usr/local/CENTRIFUGE/p_compressed anvio-gene-calls.fa -S centrifuge_
↪hits.tsv
```

5.2.10 10. Import Centrifuge data

Now import those centrifuge results for your contigs back in to anvi'o. Anvi'o can automatically read and import centrifuge output.

```
anvi-import-taxonomy-for-genes -c contigs.db -i centrifuge_report.tsv centrifuge_hits.
↪tsv -p centrifuge
```

5.2.11 11. Run single copy gene taxonomy

Now we're going to run one more thing to help us better determine the taxonomy of your MAGs.

```
anvi-run-scg-taxonomy -c contigs.db --scgs-taxonomy-data-dir /Accounts/Space_Hogs_
↪shared/anvio-scg-databases/
```

5.3 Incorporating mapping data

We've just decorated our contigs database with all kinds of things. Now, we incorporate all of the BAM files.

5.3.1 12. Import mapping files into anvi'o with anvi-profile

Now anvi'o needs to combine all of this information (your mapping, your contigs, your open reading frames, and your taxonomy) together. To do this, use anvi-profile.

-`anvi-profile` is the name of the program that combines the info together

-The `-i` flag provides the name of the sorted bam file that you copied in the step above.

-The `-T` flag sets the number of CPUs. There are 11 of you, and 96 to spare. For now, let's set it to 5 so we don't blow up the server.

-The `-M` flag sets a minimum contig length. In a project for publication, you'd want to use at least 1000, because the clustering of contigs is dependent on calculating their tetranucleotide frequencies (searching for patterns of kmers). You need to have a long enough contig to calculate these frequencies accurately. But for our purposes, let's use 500 so you can use as many contigs as possible.

```
anvi-profile -i [your sorted bam file] -c contigs.db -T 5 -M 500
```

Do this for each of your sorted bam files.

5.3.2 13. Merge them together with anvi-merge

Now merge all of these profiles together using a program called anvi-merge. You have to merge together files in directories that were created by the previous profiling step. The asterisk * is a wildcard that tells the computer, 'take all of the folders called 'PROFILE.db' from all of the directories and merge them together.'

We're also going to tell the computer not to bin these contigs automatically (called 'unsupervised' binning), we want to bin them by hand ('supervised' binning). So we use the –skip-concoct-binning flag.

This step will take a couple minutes.

```
anvi-merge */PROFILE.db -o SAMPLES-MERGED -c contigs.db
```

5.4 Visualizing and making your bins

5.4.1 14. anvi-interactive

Now the fun part with pretty pictures! Type this to open up the visualization of your contigs (of course, change the port number to the one you were assigned):

```
anvi-interactive -p SAMPLES-MERGED/PROFILE.db -c contigs.db -P 8080
```

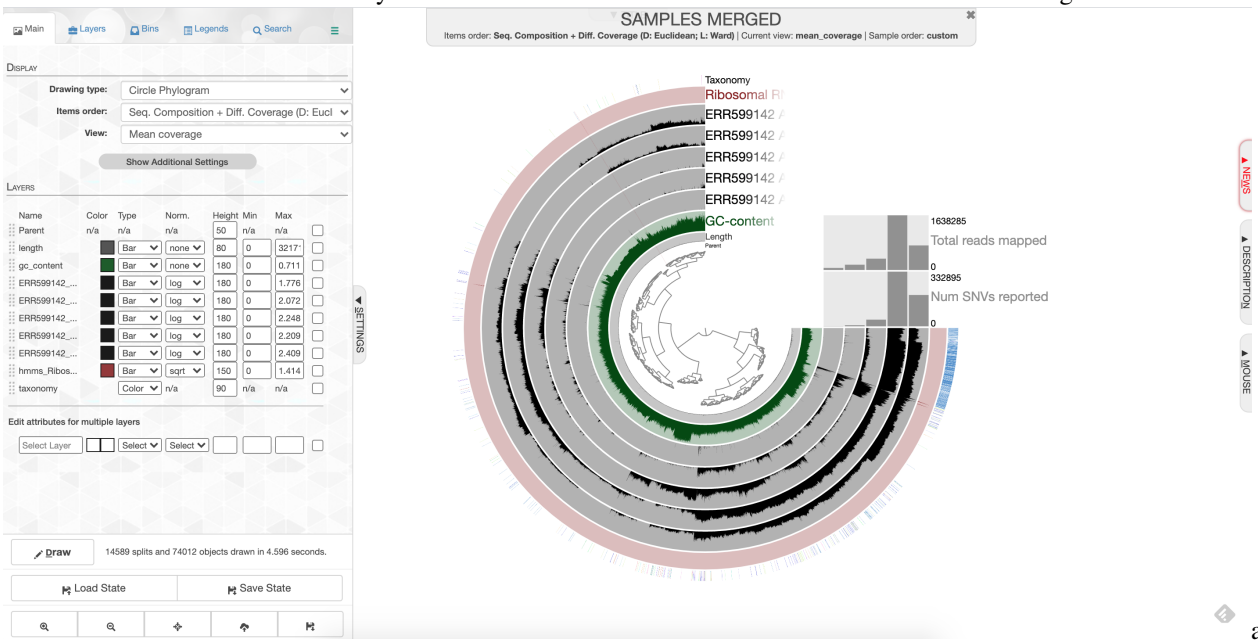
5.4.2 15. Visualize in browser

Now, open up a browser (Chrome works well) and type this into the browser window to stream the results directly from the server to your browser. NOTE that each of you will be assigned a different port number (i.e. 8080, 8081, 8082, etc); use the one you logged in with in the first step.

http://localhost:8080

Cool, eh?

Click 'Draw' to see your results! You should see something like this:



anvio

screenshot

What you are looking at:

-the tree on the inside shows the clustering of your contigs. Contigs that were more similar according to k-mer frequency and coverage clustered together.

-the rings on the outside show your samples. Each ring is a different sample. This is a visualization of the mapping that you did last week, but now we can see the mapping across the whole sample, and for all samples at once. There is one black line per contig. The taller the black line, the more mapping for that contig.

-the 'ribosomal proteins' ring shows contigs with hits to known ribosomal proteins (useful for some applications)

-the 'taxonomy' ring shows the Centrifuge designation for the taxonomy of that particular contig.

-the 'GC content' ring shows the average percent of bases that were G or C as opposed to A or T for that contig.

5.4.3 16. Make bins

We will go over the process for making bins together in class.

Because your datasets are fairly small, your bins are also going to be very small. Your percent completeness will be very low. Try to identify ~3-5 bins according to patterns in the mapping of the datasets as well as the GC content.

When you are done making your bins, be sure to click on 'Store bin collection', give it a name ('my_bins' works), and then click on 'Generate a static summary page,' click on the name of your bin collection (e.g. "my_bins"), and then click on the link it gives you. It will provide lots of information about your bins. In the boxes under the heading 'taxonomy,' you can click on the box to get a percentage rundown of how many contigs in your bin matched specific taxa according to centrifuge, if any matched.

Once you have completed your binning process, take a screenshot of your anvi'o visualization and save it as 'Figure 1.' Write a figure caption explaining what your project dataset is, and which datasets you mapped to your sample.

5.4.4 17. Finding bin information

You will find your new bin FASTA files in the directory called `~/project_directory/anvio/SAMPLES-MERGED/SUMMARY_my_bins`. I'll describe all this information below for reference; it may come in handy if you decide to use this for your final project.

-`bins_summary.txt` provides just that, with information about the taxonomy, total length, number of contigs, N50, GC content, percent complete, and percent redundancy of each of your bins. This is reflected in the summary html page you generated earlier when you clicked 'Generate a static summary page.'

If you go to the directory `bins_across_samples`, you will find information about all of your bins across all samples, such as:

-`mean_coverage.txt`, which gives the average coverage of your bins across all samples

-`variability.txt`, which gives you the number of single nucleotide variants (SNVs) per bin in each sample

If you want to know what the rest of these files mean, look here.

If you go to the directory `bin_by_bin`, you will find a series of directories, one for each bin you made. Inside each directory is a wealth of information about each bin. This includes (among other things):

-a FASTA file containing all of the contigs that comprise your bin (i.e. `Bin_1-contigs.fa`)

-a file with all of the gene sequences and gene annotations in your bin (i.e. `Bin_1-gene-calls.txt`)

-mean coverage of your bin across all of your samples (i.e. `Bin_1-mean-coverage.txt`)

-files containing copies of single-copy, universal genes found in your contigs (i.e. `Bin_1-Archaea-76-hmm-sequences.txt` and `Bin_1-Bacteria_71-hmm-sequences.txt`

-information about single nucleotide variability in your bin– the number of SNVs per kilobase pair. (i.e. `Bin_1-variability.txt`)

5.5 Analyzing your bins

OK, now the fun part! There are LOTS of things you can do with all of this information now, but for the purposes of this lab, let's do two things.

5.5.1 18. Make a heatmap showing the relative coverage of the bins in each sample

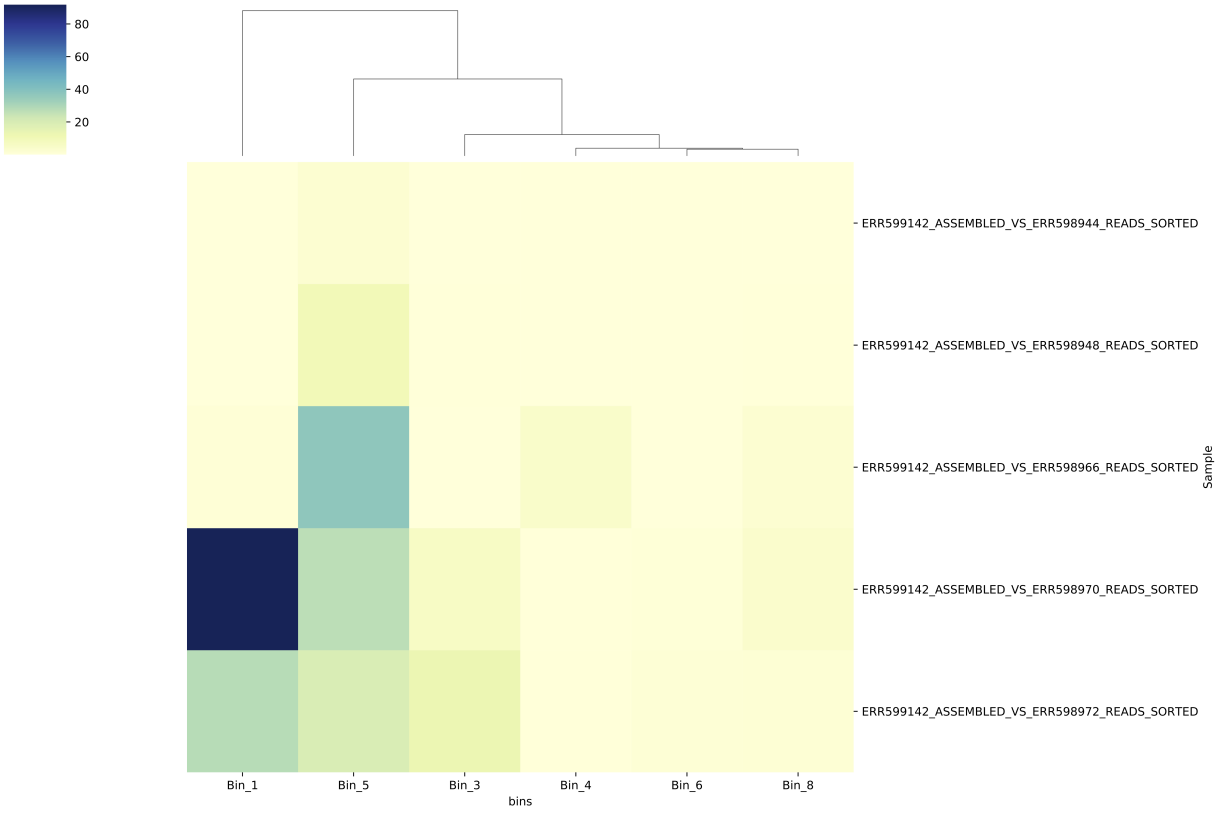
If we want to visualize how abundant each of these bins is across samples, for example to say something about which genomes might be more abundant in different places, type:

```
make_bin_coverage_heatmap.py [path to mean_coverage.txt file]
```

For example:

```
make_bin_coverage_heatmap.py SAMPLES-MERGED/SUMMARY_my_bins/bins_across_samples/mean_
↪coverage.txt
```

The script will spit out a PDF file called `bin_covg_heatmap.pdf`. Copy it over to your local computer using `scp` to take a look. You should see something that looks like this:



covg heatmap

The darker the color, the higher the coverage of that bin in that particular sample. The bins are clustered according to how similar their abundance patterns are (see the dendrogram at the top to see the clustering). The figure should give you a sense of how abundant these genomes are in different sites across the ocean.

Important side note: Most of the Python scripts you've been using are scripts that I wrote to parse and visualize these data files. I designed this class to be as open as possible and therefore I have not required you to come in with coding skills. However, for future reference, if you are thinking of going into a career in which bioinformatics is likely to be a prominent part of your daily life (as we should all strive to do!), then I strongly recommend taking a CS or stats class to formally learn something like Python or R to help with data parsing and visualization. You won't regret it!

5.5.2 19. Visualize the functional potential of each of these genomes

If you're wondering *why* these genomes are abundant in different areas– i.e., what makes these microbes different from each other that enables some to survive better in some regions? What types of metabolism do they encode? Do they have viruses inside them?– you could look at the files in `SAMPLES-MERGED/SUMMARY_my_bins/bin_by_bin/BinX/Bin_X-gene_calls.txt`, which tells you what types of genes were in that bin. If you're focusing on this for your final project, this file is where you should go for a detailed look at the gene content.

You can also get a more bird's-eye-view of gene content with a visual summary of those gene calls by doing this:

```
get_bin_functions.py [full path to the anvi'o 'bin_by_bin' directory]
```

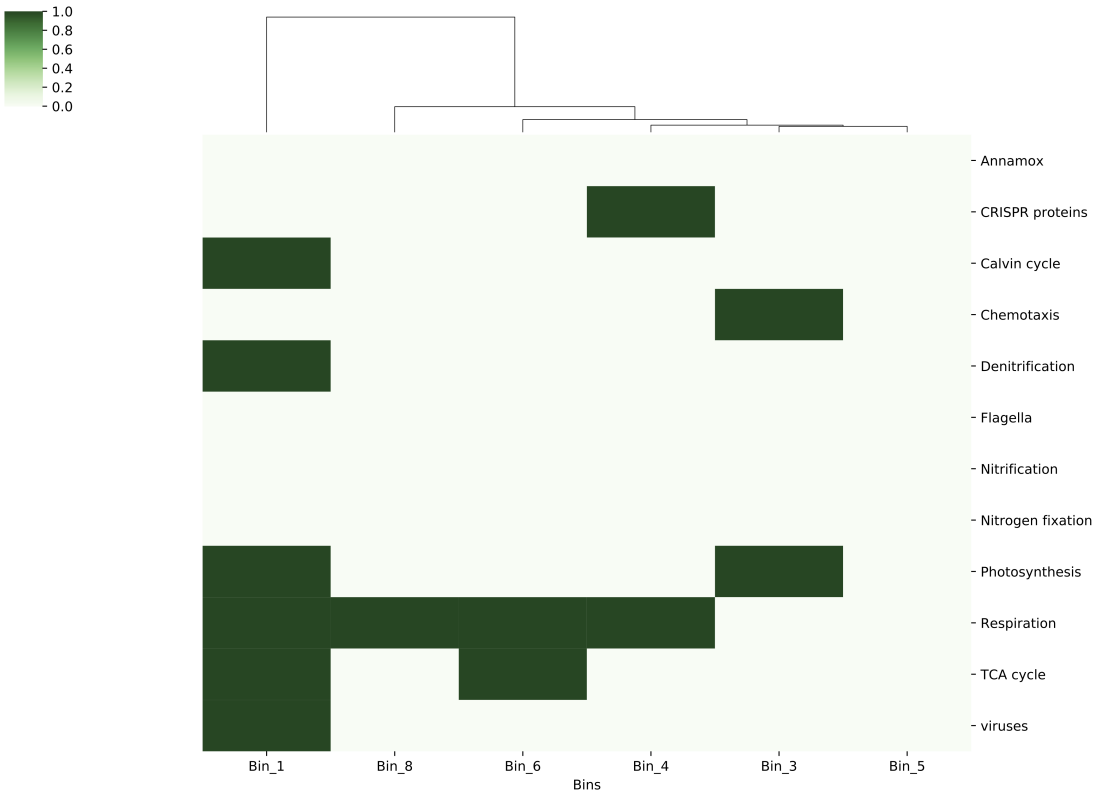
For example:

```
get_bin_functions.py /Accounts/randerson/project_directory/anvio/SAMPLES-MERGED/
↪SUMMARY_my_bins/bin_by_bin
```

This will give you an output file called `for_heatmap.txt` which you'll use in the next command. Type this:

```
make_function_heatmap.py for_heatmap.txt
```

This will create a figure called `bin_function_heatmap.pdf` that should look like this:



funct

heatmap

You've created a presence-absence grid that shows your bins at the bottom, and along the side are a bunch of gene categories. If your bin contained a gene in that category, the box for that category is colored dark green. As before, the bins are clustered according to the similarity of their functional patterns. Comparing this functions grid with your coverage heatmap might provide you with some insights (or maybe just more questions!) as to why some bins/genomes were more abundant in some regions than others.

While looking at your grid, keep in mind that these bins may not be very complete, so absence of evidence is not necessarily evidence of absence.

For this week's post-lab assignment, you won't do a mini-research question because the quality of your bins may vary, through no fault of your own. (That's real data for you...) So you get a bit of a break this week. **Please just submit the `bin_covg_heatmap.pdf` and `bin_function_heatmap.pdf` to Moodle by the start of lab next week.**

5.5.3 Final step: sharing data

Some of you might want access to each others' anvio data for your final projects. So let's share it on in our shared folder.

First, make a directory with your name on it, and then put your contigs database and SAMPLES_MERGED directory in there.

```
mkdir /Accounts/Genomics_Bioinformatics_shared/anvio_stuff/[your name]
cp contigs.db /Accounts/Genomics_Bioinformatics_shared/anvio_stuff/[your name]
cp -r SAMPLES_MERGED /Accounts/Genomics_Bioinformatics_shared/anvio_stuff/[your name]
```

CHAPTER 6

Week 6: Classifying taxonomy of short reads with mothur

6.1 Introduction

Today we're going to learn how to use sequence data to assess diversity across samples. We'll learn how to:

-classify the distribution of different types (taxa) of microbes in different datasets (i.e. what % are *Procholorococcus*, what % are SAR11, what % are Thaumarchaeota, and so on)

-calculate diversity metrics to determine which sample sites had higher richness or evenness.

Before you start, think about which datasets you'd like to compare to your own. Perhaps you want to compare a bunch of surface samples from different regions, or perhaps you want to compare the three depths from your region. This could also potentially feed right into your final project. Choose 3-5 samples to compare (including yours).

6.2 Using mothur to profile the taxonomy of your dataset

6.2.1 1. Log in

Once you have decided on what samples to compare, log on to baross using ssh on your Terminal. Then make a new directory and change directory into it.

```
mkdir ~/project_directory/taxonomy
cd project_directory/taxonomy
```

6.2.2 2. Copy over data

The Tara Ocean people have already identified all of the reads that matched 16S ribosomal RNA from their metagenomes, and they made separate FASTA files with just those reads. (Thanks, French scientists!) I copied those FASTA files into the directories listed in `/usr/local/data/Tara_datasets/`. Copy over all of the relevant 16S rRNA files that you will need into your new taxonomy folder.

```
cp /usr/local/data/Tara_datasets/[project sample site of interest]/[16S rRNA data␣
↪file of interest] .
```

For example, you might do something like this:

```
cp /usr/local/data/Tara_datasets/Arabian_Sea/ERR598966_MERGED_FASTQ_16SrRNA_10000.
↪fasta .
cp /usr/local/data/Tara_datasets/coastal_South_Africa/ERR598972_MERGED_FASTQ_16SrRNA.
↪fasta .
cp /usr/local/data/Tara_datasets/North_Pacific/ERR598995_MERGED_FASTQ_16SrRNA_10000.
↪fasta .
```

Hot tip! If you want to copy over all of the 16S rRNA datasets from one folder, you could use the asterisk (wildcard). For example: `cp /usr/local/data/Tara_datasets/North_Pacific/*16S* .` That would copy over all of the files containing `16S` in their title in the North Pacific folder.

6.2.3 3. Open mothur

Now, we're going to use a program called `mothur` to analyze these sequences. mothur is a bit different from other programs that we've used in that we can enter a mothur interface and type commands that are specific to mothur. To enter the mothur program, simply type this:

```
mothur
```

6.2.4 4. Create groups file

Right now, the 16S rRNA-matching reads from each sample site are in separate files. Pretty soon we are going to merge all of your FASTA files together in order to compare them. Before we do that, we need to tell mothur how to tell them apart once they are merged. We do that with the make.group command. Replace the file names below with your 16S rRNA fasta file names. Substitute the group names with a suitable name for your sample so that you can recognize it, like `NPacific_DCM` or `Arabian_surface`.

Note: any time you see something in [brackets] in a command, it means you have to substitute that with your own data. Don't include the brackets in your command.

```
make.group(fasta=[file1.fasta]-[file2.fasta]-[file3.fasta], groups=[group1]-[group2]-
↪[group3])
```

For example:

```
make.group(fasta=ERR598944_MERGED_FASTQ_16SrRNA_10000.fasta-ERR599001_MERGED_FASTQ_
↪16SrRNA_10000.fasta-ERR599078_MERGED_FASTQ_16SrRNA_10000.fasta, groups=meso-
↪transect-surface)
```

6.2.5 5. Look at the groups file

This command should generate a file that is called `groups`. Take a look at it with `less`.

Hot tip #2! You can use the command system() if you want to use Unix commands while you are using mothur.

You will see that each sequence name is linked up with the group name that you provided. That way mothur can combine all of the sequences together into one file, but you can still keep track of which one belongs to which sample. This file will be essential for allowing mothur to compare your samples later on.

```
system(less groups)
```

6.2.6 6. Merge FASTA files together

Now you can merge all of your FASTA files together, and the .groups file will record which sequences came from which file. The output here will be a file called `merged.fa`. Again, substitute "file-1.fa" and so on with the names of your 16S rRNA fasta files.

Hot tip #3! use the up arrow on your keyboard to call up the last command you typed, and then edit that command instead of retyping all of the filenames again.

```
merge.files(input=[file1.fa]-[file2.fa]-[file3.fa], output=merged.fa)
```

For example:

```
merge.files(input=ERR598944_MERGED_FASTQ_16SrRNA_10000.fasta-ERR599001_MERGED_FASTQ_
↪16SrRNA_10000.fasta-ERR599078_MERGED_FASTQ_16SrRNA_10000.fasta, output=merged.fa)
```

7. Classify your sequences

Everything we've done so far is basically record-keeping. Now it's time to do SCIENCE!

We will classify our sequences by comparing them to a reference database. We will the SILVA database to compare these sequences. (It's a very good, well-curated 16S rRNA database.)

Note! You will see lots of warnings along the lines of: "[WARNING]: xxx could not be classified." We are going to have to leave these sequences out of the analysis! This means all of the unknowns will be grouped together even though they most likely represent many different species, so they will be missing from our diversity analyses later on. This is why OTUs are often a better way to go here, but unfortunately our metagenomic data is too messy to be able to make nice OTUs.

```
classify.seqs(fasta=merged.fa, group=groups, reference=/usr/local/data/silva_
↪databases/silva.seed_v119.align, taxonomy=/usr/local/data/silva_databases/silva.
↪seed_v119.tax)
```

8. Open classified sequences

Use scp to transfer your files over to your local desktop. Open the file that is called `merged.seed_v119.wang.tax.summary` in Excel. (You may have to change the name so the file ends in '.txt' or Excel won't recognize it as a valid file to open.) Here is the definition of the columns, from left to right:

-Taxonomic level is in the farthest left-hand column. The lower the number, the larger the phylogenetic classification, starting with domain, then phylum, class, order, family, genus, species. For example, Archaea, Bacteria, Eukarya, and 'unknown' are taxonomic level 1. Taxonomic level 2 classifies different phyla of Archaea, Bacteria, and Eukaryotes. Taxonomic level 3 classifies different classes of those phyla, and so on.

-The rankID provides a means of keeping track of where that particular organism falls. For example, the SAR_11 clade is rankID 0.2.17.2.9, which means it is a clade within the Alphaproteobacteria (0.2.17.2), which are a clade within the Proteobacteria (rankID 0.2.17), which is a clade within the Bacteria (rankID 0.2).

-The taxon column tells you the name of the taxon.

-The daughter level tells you how may levels down you are in the phylogeny.

-The 'total' tells you how many total sequences are within that taxonomic category.

-Each of the following columns gives you the taxonomic breakdown for that sample.

I recommend sorting your Excel spreadsheet by taxlevel. Make either a pie chart or stacked bar chart for each sample depth based on the taxonomic distribution at taxonomic level 2 to compare the taxonomic breakdown between each of your samples. **Save these charts as Figure 1(abc) and provide a figure caption for this figure.**

9. Share your data

Your classmates may wish to use your taxonomy data for their project datasets. Please rename your taxonomy files and share them on the class_shared directory.

```
mv merged.seed_v119.wang.tax.summary [newname]
cp [newname] /Accounts/Genomics_Bioinformatics_shared/taxonomy
```

For example:

```
mv merged.seed_v119.wang.tax.summary rikas_taxonomy.summary
cp rikas_taxonomy.summary /Accounts/Genomics_Bioinformatics_shared/taxonomy
```

10. Calculate the diversity of your sample

Now we're going to calculate the diversity at each of your sample sites. These will be pure calculations in Excel rather than on the command-line. mothur can do this using OTUs, but our data is too messy to create nice OTUs!

We'll calculate diversity using two measures: species richness (r) and the Shannon-Weiner Index (H'). The Shannon-Weiner Index (H') is meant to take into account both the taxon richness (i.e. how many different taxa there are) and evenness (i.e. does one taxon dominate, or are they evenly distributed?)

We are going to calculate the diversity of each of your sample sites at taxonomic level 2. This means that each entry at level 2 (i.e. 'Euryarchaeota,' 'Thaumarchaeota,' 'Proteobacteria') will count as one taxon. Calculate the **richness** and the **Shannon-Weiner index** for each of your samples.

Richness = r = number of taxa in your sample

The equation for the Shannon-Weiner index is:

H' = -Σ (Pi ln(Pi))

H' = index of taxonomic diversity, the Shannon-Weiner Index

Pi = proportion (percent) of total sample belonging to the ith taxon

ln = natural log (log base e = not the same as log!)

This index takes into account not just the number of taxa (richness) in a sample, but also how evenly distributed the taxa are (evenness) within the sample. The index increases by having more richness and/or by having greater evenness.

Hint: -Σ (Pi ln(Pi)) = -((Ptaxon1*ln(Ptaxon1)) + (Ptaxon2*ln(Ptaxon2)) + (Ptaxon3*ln(Ptaxon3)) + . . .)

I suggest that you start by calculating the total number of sequences for each sample site for all of taxonomic level 2. Then, for each taxon within taxlevel2, calculate the total proportion of sequences belonging to that taxon (Pi). Then calculate H'.

Excel command for natural log: LN() Note that in Excel, LN(0) = error, so skip the cells with a value of 0. Pay attention to your use of parentheses!

Please do these calculations in a way that is clear so that I can track your calculations. You will be submitting these Excel spreadsheets as part of your lab assignment for this week. Please compile the total number of sequences, the species richness, and the Shannon-Weiner index for each of your sample depths in a table. **Save this as Table 1 and provide a caption.**

11. Compare diversity with metadata

In your Excel spreadsheet, make 5 columns containing metadata for each of your samples (1 column each for temperature, chlorophyll, nitrate, oxygen, and salinity). Make a scatterplot for each of the metadata versus your Shannon-Weiner index (H') for each of the samples. For example, one plot will have temperature on the x axis, and H' on the y axis, one will have chlorophyll on the x axis, and H' on the y axis, and so on. For each plot, plot a trendline through the data and include the equation and the R-squared value. (Note that since in most cases you won't have more than, say, 3 samples, we're not going to do a formal statistical test on these datasets. Therefore, based on what you've done here, we can't yet conclude whether the relationships are significant.) **Save these 5 plots as Figure 2(abcde).**

12. Post-lab assignment

Check for understanding questions:

Q1. Many of your sequences were unclassifiable. How would this likely affect your richness calculations for each sample? Explain why.

Q2. What is the difference between richness and the Shannon-Weiner index? Describe a situation in which you might have a high richness but a relatively low Shannon-Weiner index.

Q3. Does your taxonomic diversity, as calculated by the Shannon-Weiner index, correlate with any of the metadata for your sample (temperature, chlorophyll, nitrate, oxygen, salinity)? (The R squared value should vary between 0 and 1; the stronger the correlation, the closer the R-squared value is to 1. We did not calculate p-values or conduct a more rigorous statistical analysis, but the R-squared value will tell you how closely the variables are correlated.) Write a short paragraph speculating on any correlations you find. (It's possible the correlations will be terrible.)

Ecological connection question:

Look back at the 3-5 samples that you selected. What kinds of trends do you see differentiating them? What were you expecting to see, and do your results support your initial expectations? Write 1-2 paragraphs explaining your results in light of what we learned in class about marine taxa and diversity in various ocean basins and at different depths.

Summary of what to turn in next week: For this week's post-lab assignment, please submit the following (should all be in the same document):

-Figure 1(abc)

-Figure 2(abcde)

-Table 1

-The answers to the three "check for understanding" questions above

-The response to the ecological connection question

-Your Excel spreadsheet so that I can check your calculations (submitted separately)

CHAPTER 7

General Stuff

7.1 ssh-ing into liverpool / baross

Liverpool:

```
ssh [username]@liverpool.its.carleton.edu
```

Baross

```
ssh [username ]@baross.its.carleton.edu
```

7.2 Using screen/ tmux

(We have been using screen in class.)

See: Cheat Sheet

^ = control key

```
| Action                      | tmux          | screen     |
|-----------------------------|---------------|------------|
| start new session           | `tmux`        | `screen`   |
| detach from current session| `^b d`        | `^a ^d`    |
| re-attach detached session | `tmux attach`| `screen-r` |
| list sessions               | `^b s`        | `screen-r` |
```

7.3 File Transfer

7.3.1 Filezilla

- Open Filezilla
- Host: sftp://liverpool.its.carleton.edu
- Username: Carleton username
- Password: Carleton password
- Port: 22
- Click QuickConnect

7.3.2 Secure Copy

From baross to local computer:

```
scp [username]@baross.its.carleton.edu:/Accounts/[username]/[path of your destination␣
↪directory]/[some_file.txt] ~/Desktop
```

From local computer to baross:

```
scp ~/Desktop/[some_file.txt] [username]@baross.its.carleton.edu:/Accounts/[username]/
↪[path of your destination directory]
```

CHAPTER 8

Bioinformatics Tools

Summary of purpose & usage of bioinformatics tools seen in class.

8.1 IDBA-UD

Assembles reads into contigs

Example usage (replace names in brackets with your own names and remove brackets):

```
idba_ud -r [reads file] -o [output folder name]
```

- The `-r` gives it the "path" (directions) to the reads for your reads. `../` means it is in the directory outside of the one you're in.
- The `-o` flag tells the program what you want the output directory to be called.

8.2 Prokka

Finds ORFs in assembled contigs and annotates them

Example usage (replace names in brackets with your own names and remove brackets):

```
prokka [assembled contigs fasta file] --outdir [name of output directory]
```

To get a summary of how many ORFs were assigned to different COG categories based on your Prokka data, do this:

```
get_ORF_COG_categories.py [name of your Prokka tsv file]
```

8.3 BLAST

Find a match to a sequence in your own sequence file, not the NCBI database

Example usage (replace names in brackets with your own names and remove brackets):

First, make a BLAST database:

```
makeblastdb -in [example sequence file] -dbtype [prot or nucl]
```

- -in gives the name of the file you want to BLAST against.
- for dbtype, use `prot` for an amino acid file and `nucl` for a nucleotide file.

Then, do your BLAST.

8.3.1 blastp

```
blastp -query [example query file] -db [example database file] -evalue 1e-05 -outfmt␣
↪6 -out [example_output.blastp]
```

- `blastp` does a protein vs protein blast. (other choices are `blastn`, `blastx`, `tblastn`, `tblastx`.)
- `-query` defines your blast query– in this case, the Pfam seed sequences for the CRISPR RAMP proteins.
- `-db` defines your database– in this case, the toy assembly ORFs.
- `-evalue` defines your maximum e-value, in this case 1x10-5
- `-outfmt` defines the output format of your blast results. option 6 is common; you can check out https://www.ncbi.nlm.nih.gov/books/NBK279675/ for other options.
- `-out` defines the name of your output file. I like to title mine with the general format `query_vs_db.blastp` or something similar.

8.3.2 tblastn

```
tblastn -query [example query file] -db [example database file] -evalue 1e-05 -outfmt␣
↪6 -out [example_output.tblastn]
```

- `tblastn` does a protein vs translated nucleotide blast. (other choices are `blastn`, `blastx`, `blastp`, `tblastx`.)
- `-query` defines your blast query– in this case, the Pfam seed sequences for the CRISPR RAMP proteins.
- `-db` defines your database– in this case, the toy assembly ORFs.
- `-evalue` defines your maximum e-value, in this case 1x10-5
- `-outfmt` defines the output format of your blast results. option 6 is common; you can check out https://www.ncbi.nlm.nih.gov/books/NBK279675/ for other options.
- `-out` defines the name of your output file. I like to title mine with the general format `query_vs_db.blastp` or something similar.

8.4 Making a tree

First make an alignment with muscle, then make a tree with RAxML

First, use nano or some other text editing tool to make a fasta file with your sequences of interest. The following is an example of how to make a tree using amino acid (not nucleotide) sequences.

Example usage (replace names in brackets with your own names and remove brackets):

```
muscle -in [example_sequence_file.fasta] --out [example_sequence_file_aligned.afa]
convert_afa_to_phy.py [example_sequence_file_aligned.afa]
raxmlHPC-PTHREADS-AVX -f a -# 20 -m PROTGAMMAAUTO -p 12345 -x 12345 -s [example_
↪sequence_file_aligned.phy] -n [example_tree_name.tree] -T 4
```

Open the RAxML_bipartitions.example_tree_name.tree file in ITOL.

8.5 Mapping

Map short reads against a reference

Example usage (replace names in brackets with your own names and remove brackets):

```
bowtie2-build [reference fasta file] [reference_file.btindex]
bowtie2 -x [reference_file.btindex] -f -U [reads_to_map.fasta] -S [output_file.sam]
samtools view -bS [output_file.sam] > [output_file.bam]
samtools sort [output_file.bam] -o [output_file_sorted.bam]
samtools index [output_file_sorted.bam]
```

To get ORF coverage:

```
make_bed_file_from_gff_file_prokka.py [your gff file from prokka]
samtools bedcov [your bed file] [your sorted bam file] > [ an output file that ends␣
↪in _ORF_coverage.txt]
```

8.6 anvi'o

Make bins from metagenomic assemblies and BAM files

The protocol for week 6 laid out anvi'o step by step. The anvi'o tutorial is another good source of information.

8.7 mothur

Analyze 16S data to find taxonomy, OTUS

The protocol for week 7 laid out mothur step by step. The mothur wiki is another good source of information.

CHAPTER 9

Legacy Items

9.1 Prodigal

Finds ORFs in assembly file

Example usage:

```
mkdir ORF_finding
cd ORF_finding
prodigal -i ../toy_assembly/toy_dataset_assembly_subsample.fa -o toy_assembly_ORFs.
↪gbk -a toy_assembly_ORFs.faa -p single
```

- The `-i` flag gives the input file, which is the assembly you just made.
- The `-o` flag gives the output file in Genbank format
- The '-a" flag gives the output file in fasta format
- The `-p` flag states which procedure you're using: whether this is a single genome or a metagenome. This toy dataset is > a single genome so we are using –p single, but for your project dataset, you will use –p meta.

9.2 Interproscan

ORF Fasta File -> Annotated TSV

Used to efficiently and effectively annotate proteins. Compares your open reading frames against several protein databases and looks for protein "signatures," or regions that are highly conserved among proteins, and uses that to annotate your open reading frames. It will do this for every single open reading frame in your dataset, if it can find a match.

Example usage:

```
interproscan.sh -i toy_assembly_ORFs.noasterisks.faa -f tsv
```

- The `-i` flag gives the input file, which is your file with ORF sequences identified by Prodigal, with the asterisks removed.
- The `-f` flag tells Interproscan that the format you want is a tab-separated vars, or "tsv," file.

CHAPTER 10

Contributing

This is a sphinx project, which generates documentation for Carleton College BIOL.338 Genomics & Bioinformatics.

To contribute, you must:

1. Be able to edit markdown (easy!)
2. Be able to run sphinx, to generate webpages from markdown (easy!)
3. Be able to use git to upload file changes to github (also easy!)

How do these tools fit together?

- We write the docs in markdown, then use sphinx to render beatutiful webpages from them.
- We use Git/Github to store the current version of the project online.
- We can also use ReadTheDocs to automatically build/serve our website, based on the current files stored on Github!

10.1 Step 1: Markdown

Markdown is a fairly simple specification for formatting text.

Markdown lets you indicate that things should be bolded or italicized or seen as headers or links in "plain-text" (ie, by typing certain characters rather than pushing application-specific buttons).

You can even write $\LaTeX$ in Markdown, by enclosing your Latex stuff in `$`.

> For syntax reference: see commonmark.org/help. (Though keep in mind some Markdown rendering "engines", like the one on GitHub, support slightly different syntax, beyond and even different than the syntax specified by CommonMark.)

10.1.1 Using Markdown

A nice way to write Markdown and see it rendered into a formatted documnt is to use the Atom text editor, with the fantastic extension Markdown Preview Enhanced.

10.2 Step 2: Sphinx

10.2.1 Installing Sphinx

1. Install sphinx: `pip install sphinx`
2. Install the markdown extension: `pip install recommonmark`
3. Install the ReadTheDocs Sphinx theme: `pip install sphinx_rtd_theme`

   Note: `pip` commands are run at the command line / in Terminal.

10.2.2 Using Sphinx

Prepare materials:

- Write your markdown documents and put them somewhere, like the `pages/` folder.
- To control which files are included: edit the file `index.rst`
- To alter other settings: edit the file `conf.py`

Build the website:

- Run `make clean` in terminal to empty out the `_build/` directory
- Run `make html` in terminal to generate html from the markdown files.
- Type `open _build/html/index.html` in terminal to view your website with your browser (or open that file using Finder.)

  Note: A good example project, for inspiration on using sphinx/markdown is the Requests package, which is on Github. Look in their `docs/` directory, find files like `index.rst` and press the `raw` button.

10.3 Step 3: Github

When you're ready to push your changes to github.

10.3.1 With Permissions

If you have permissions to push to the Github repository, commiting and pushing your changes just requires a few lines:

```
# see what has changed
git status

# stage modified files
git add -u
```

(continues on next page)

(continued from previous page)

```
# add any new files
git add <filename> <filename>

# commit changes
git commit -m "< write a msg like 'modify protocol 5'>"

# push changes to remote server (github)
git push
```

10.3.2 Without Permissions

If you don't have Github pushing permission for this project, you should fork the project, commit your changes, then open a pull request. This is the way open source software gets built. Hooray for open source!!!

Here's a good resource on this workflow: https://gist.github.com/Chaser324/ce0505fbed06b947d962

CHAPTER 11

Authors

- Rika Anderson - Created/wrote all labs
- Dustin Michels, dustin@dustinmichels.com - Helped port docs to markdown