Carleton Bioinformatics Documentation

Rika Anderson

Sep 08, 2023

Protocols:

CHAPTER 1

Week 3: Introduction to Unix/Linux and the Server; Assembly with IDBA-UD; ORF Calling and Annotation with Prokka

Rika Anderson, Carleton College

1.1 Connecting to baross

1.1.1 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. (In case anybody cares, baross is named after this absolute legend of a scientist, who happened to be my PhD advisor.) baross (the server) 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. Some Windows machines have a native Terminal now. If you don't have one, I recommend installing a Ubuntu terminal. If that doesn't work, you can use PuTTY.. If all else fails, ask Jimmy. (Or do that first.)

1.1.2 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.

1.1.3 3. ssh

We will use something called ssh, or a secure socket shell, to remotely connect to another computer (in this case it will be 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
```

1.1.4 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.

1.1.5 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
```

1.1.6 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.

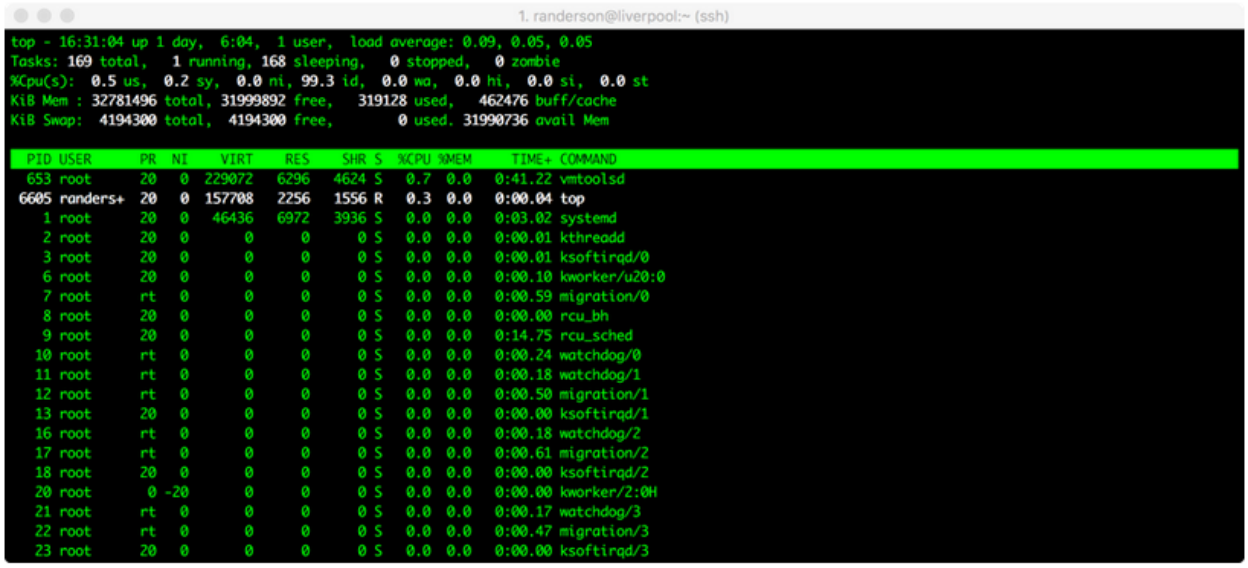
1.2 Important things to know about when using a remote shared server

1.2.1 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. Please also keep an eye on the memory (RAM) usage (where it says %MEM in top) and ensure we're not using more than 80% of total RAM across all jobs.

To quit out of top, type: `q`

1.2.2 8. screen

Sometimes we will run processes that are so computationally demanding that they will run for several hours or overnight. 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.

1.3 Creating an assembly and evaluating it (toy dataset)

1.3.1 9. mkdir

Let's say we've 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
```

1.3.2 10. cd

Change directory into your toy dataset directory:

```
cd toy_dataset_directory
```

1.3.3 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 .
```

1.3.4 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
```

1.3.5 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
```

1.3.6 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
```

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. 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.

1.3.7 15. Examine output

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

`less scaffold.fa`

1.3.8 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`.

1.3.9 17. Evaluate assembly quality

Now that you have an assembly, we're going to evaluate its quality. To do that, 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.)

```
cd ~/toy_dataset_directory/toy_assembly
quast.py contig-20.fa contig-40.fa contig-60.fa contig-80.fa contig-100.fa scaffold.fa
```

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.

Quast called the output `quast_results` by default. You can change that using the `mv` command: `mv quast_results toy_assembly_quast_evaluation`

1.3.10 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.

Open up another Terminal window on your own computer. Don't ssh into anything. Navigate to your home directory.

```
cd ~
```

Type this (substituting your own username below):

```
scp -r username@baross.its.carleton.edu:~/toy_dataset_directory/toy_assembly/toy_
↪assembly_quast_evaluation .
```

That means: copy something from the remote server baross and put it right here on my local computer.

Find your newly copied folder on your own computer. Double-click on the file called "report.html" and discuss it with your group (below).

1.3.11 19. Pause to check for understanding

Take a pause here and check in with the other folks at their table. Help them catch up if they aren't at this step yet. When you're all at this point, discuss the following questions. You'll submit these as part of your postlab writeup described at the end of this protocol document (write your group members' names on the writeup as well).

- a) Examine the Quast output. Describe and explain the pattern you observe in terms of N50 as the kmer size increases (from 'contig-20.fa' all the way to 'contig-100.fa').
- b) Examine the Quast output. How does the N50 of your scaffold file compare to your contig-100 file? Explain why.
- c) Examine the Quast output. Which assembly output file had the most contigs? The fewest? Explain why.

1.4 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 works 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
```

1.4.1 20. Run Prokka

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

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

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

1.4.2 21. 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_01252021.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. And the file that ends in `.gbk` is one 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.

1.4.3 22. 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_09252020.tsv .
```

1.4.4 23. 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.

This document can be very useful: now you can see ALL of the ORFs in your metagenome and their putative annotation! This file can be handy when you're working on your postlab and/or when you're working on your final project.

1.5 Create and evaluate assembly (project dataset)

Cool! Now that you've learned how to assemble and annotate your toy datasets, now we're going to start working on your project datasets. Before we do that, I encourage you all to start a **computational lab notebook**. I like to use a text editor like BBEdit or something similar and save it on Google Drive or something. In that notebook, it will be crucial that you keep a record of all the commands you run 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.

1.5.1 24. 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
```

1.5.2 25. Copy project dataset

Next, copy your project dataset into that directory. Your assigned project dataset is listed in a Google Sheets 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 .
```

1.5.3 26. Start assembly (except not really)

OK, so normally, I'd tell you to start your assembly using IDBA-UD. You would run a command like this:

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

BUT DON'T DO IT!! Instead, I've pre-assembled your project datasets for you, because otherwise you'd be sitting here for hours. Copy them into your directory like this:

```
cp -r /workspace/data/Genomics_Bioinformatics_shared/assemblies/[your assembly name] .
```

Look inside using `ls`. Inside, you will find the same files you saw with the toy dataset: contig-20, contig-40, contig-60, contig-80, and contig-100. However, the Tara Oceans folks provided single-end reads instead of paired-end reads, which 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`.

1.5.4 27. 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. Do this while you are inside the directory with your assembly files:

project dataset (substitute your own assembly name):

```
cd ~/project_directory/[your assembly directory]
anvi-script-reformat-fasta contig-100.fa -o ../ERR598966_assembly_reformatted.fa -l 0␣
↪--simplify-names
```

1.6 Annotate your own project datasets

Now you're going to annotate your own project dataset assemblies using `Prokka`.

1.6.1 28. Run Prokka on your project dataset

If you aren't in your project directory right now, move into it and start `prokka`. You're should probably open a screen session, since this might take a while. Run the following, again, substituting your own project dataset name.

```
screen -S prokka
prokka ERR598966_assembly_reformatted.fa --outdir prokka_project
Ctrl+A d
```

1.7 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.

1.7.1 34. 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
```

1.7.2 35. 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]
```

Et 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`. So now, you have a detailed list of all of the ORFs and their functions (the original Prokka .tsv file) AND a list of what general categories your ORFs fall into (the new 'cog_categories.txt' file). Imagine the possibilities!

1.7.3 36. Share your data

Please copy the data you generated today and put them into the folder that we'll be sharing as a class, substituting the placeholders below with the name of your own file. This will be the shared folder for all of the data that you generate as a class.

Copy your assemblies to the shared class assemblies folder:

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

Copy your Prokka tsv and faa files to the shared class Prokka folder, and rename it with the name of your project dataset so it's recognizable. for example:

```
cp PROKKA_09222020.tsv /Accounts/Genomics_Bioinformatics_shared/PROKKA_results/
↪ERR598995_ORFs.tsv
cp PROKKA_09222020.faa /Accounts/Genomics_Bioinformatics_shared/PROKKA_results/
↪ERR598995_ORFs.faa
```

Copy your COG categories file into the shared class Prokka folder, and change the name so it's easily recognizable. For example:

```
cp PROKKA_09222020_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.

1.7.4 37. 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
```

1.8 Lab assignment this week

Write a mini research question based on your ORF annotations. If you like, you can compare your sample to someone else's. This doesn't have to be overly in-depth: ask a simple question that can be answered by the files you and/or your classmates have generated today.

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?"

Or for example, you might ask "What genes are present in the mesopelagic zones that are missing from the surface ocean?"

Or, "Are there more genes labeled as "hypothetical" genes in the Antarctic Ocean compared to the North Pacific?"

Or, "Within a single sample, are there more genes related to energy metabolism or replication/recombination/repair?"

Make a plot or table that illustrates 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, and what else you might want to investigate related to this idea in the future.

Submit this plus your "check for understanding" questions above (in a single document) on the class Moodle page by lab time next week. (Note: your "check for understanding" questions come from group discussion; your mini research question should be done independently, but you're free to talk to each other for ideas or help troubleshooting. The final product should represent your own work.)

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 2

Week 4: Local alignments and sequence search with BLAST, global alignments with MUSCLE, and making trees with RAxML

Rika Anderson, Carleton College

2.1 Logging in to the remote server

2.1.1 1. Login

Boot as a Mac on the lab computer.

2.1.2 2. Terminal

Find and open the Terminal application (`Applications/Utilities`). (For future reference if you're doing this from home, if you're on a PC , open your Ubuntu terminal or your PuTTY terminal.)

2.1.3 3. Connect to baross

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

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

2.2 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.

2.2.1 4. Copy a mystery sequence

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.

Here's a mystery gene. Let's BLAST it. First, copy this sequence below.

```
>mystery_gene
MVPQTETKAGAGFKAGVKDYRLTYYTPDYVVRDTDILAAFRMTPQLGVPPEECGAAVAAESSTGTWTTVW
TDGLTSLDRYKGRCYDIEPVPGEDNQYIAYVAYPIDLFEEGSVTNMFTSIVGNVFGFKALRALRLEDLRI
PPAYVKTFVGPPHGIQVERDKLNKYGRGLLGCTIKPKLGLSAKNYGRAVYECLRGGLDFTKDDENVNSQP
FMRWRDRFLFVAEAIYKAQAETGEVKGHYLNATAGTCEEMMKRAVCAKELGVPIIMHDYLTGGFTANTSL
AIYCRDNGLLLHIHRAMHAVIDRQRNHGIHFRVLAKALRMSGGDHLHSGTVVGKLEGEREVTLGFVDLMR
DDYVEKDRSRGIYFTQDWCSMPGVMPVASGGIHVWHMPALVEIFGDDACLQFGGGTLGHPWGNAPGAAAN
RVALEACTQARNEGRDLAREGGDVIRSACKWSPELAAACEVWKEIKFEFDTIDKL
```

2.2.2 5. Navigate to NCBI site

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

2.2.3 6. Paste sequence and BLAST

Paste 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.

2.2.4 7. Run tblastn

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.)

2.2.5 8. Pause to check for understanding with your lab group

Take a pause here and check in with the others at your table. If others haven't quite caught up to you yet, help them catch up. As a table, discuss and answer the following questions. Include the responses as part of your postlab assignment.

Check for understanding:

Q1. First look at your blastp results. Take a look at the list of hits below the top hit. Why do you think there are so many different hits with a similar function?

Q2. Describe the difference in your `tblastn` and `blastp` results. Why do they look different? How does this reflect the databases you are BLASTing against? Discuss and explain in what scenarios you might choose to use one over the other.

2.2.6 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 "3064 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.

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. (Adjust the name of your PROKKA file if you need to.) The following command illustrates some common parameters.

```
blastp -query PF03787_seed.txt -db PROKKA_01252021.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.

As we discussed in class, the e-value is a way to establish a cutoff of what makes a "good" BLAST hit. Smaller e-value = better hit.

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

18. 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
```

18. Post-lab questions

As before, take a pause and check in with your lab group. Help them catch up. As a group, take a look at your BLAST results and answer the following questions for the postlab.

Check for understanding:

Q3. (a-e)

- Which protein among your Pfam query sequences had the best hit?
- What was the percent identity?
- Which of your ORFs did it match?
- Does this ORF have hits to other sequences within your query file? What do you think this means?
- Similarly, did multiple ORFs in your dataset have hits? What do you think this means?

19. Post-lab questions

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
```

With your group, discuss and answer the following question for the postlab:

Check for understanding:

Q4. How do these BLAST results differ from your previous BLAST? Explain why.

2.3 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.

2.3.1 20. 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. Today we're just going to align 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 .
```

Take a look at it with `less`. It's just a FASTA file with a bunch of sequences.

2.3.2 21. 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."

2.3.3 22. 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 the lab computer. 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]
```

2.3.4 23. 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.)

2.4 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.

2.4.1 24. To make your tree, type this:

```
raxmlHPC-PTHREADS-AVX -f a -# 20 -m PROTGAMMAAUTO -p 12345 -x 12345 -s toy_dataset_
↪PSII_protein_aligned.afa -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 aligned FASTA 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.

2.4.2 25. 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. `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.

2.4.3 26. Visualizing the tree

Copy your Newick file (RAxML_bipartitions.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. IF you wanted to save a picture of your tree, you could click on the "Export" tab, choose the PDF format, and click "Export." It should pop up in a new tab.

2.5 Post-lab assignment

For your post-lab assignment, compile the 4 "check for understanding" questions and answer the critical thinking question below.

Critical thinking question:

As with last week, develop a simple question about your project dataset that you can answer using BLAST, making trees, or both.

For example, your question could be something like: 'I found a cytochrome c gene in my dataset. Where does it fit on a tree of other cytochrome C genes from Pfam?' (Make a tree)

Or, "How many ORFs in my dataset have a match to a viral gene? How does that compare to a dataset from a different depth from the same part of the ocean?" (Use BLAST)

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: concisely include all the important details needed to repeat what you did, like 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. Show a table or a figure if appropriate.

-What do you think this tells 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.)

Submit via 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 5: Mapping with bowtie2

3.1 1. Log in to the remote server

Boot your computer as a Mac and use the Terminal to ssh in to baross.

3.2 Mapping with a toy dataset

3.2.1 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*) 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 toy dataset directory, make a new directory called "mapping," then change into that directory.

```
cd ~/toy_dataset_directory
mkdir mapping
cd mapping
```

3.2.2 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 .
```

3.2.3 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.

3.2.4 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 read about in the Zimmer "Game of Genomes" articles.

- `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
```

3.2.5 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.

3.2.6 7. samtools

Now you will use a package called `samtools` to convert the SAM file into a non-human-readable BAM file. Carl Zimmer spent a lot of time trying to obtain his BAM files in that "Game of Genomes" article– now you get to make one yourself! (But, uh, not of your own genome, obviously. . .)

```
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 sam file to a bam file. (Bioinformatics jokes = still not very funny.)

3.2.7 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.

3.2.8 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.

3.2.9 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.

3.2.10 11. Copy to local computer

Now we're going to visualize this. Copy the entire "mapping" folder over to your local computer using scp (or Filezilla if you prefer.)

```
scp -r username@baross.its.carleton.edu:~/mapping/ .
```

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 in whatever folder you happen to be in. You could tell the computer to put it on the Desktop or wherever you like by simply providing the path instead of a period in the second part of the command.

3.2.11 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 (`toy_dataset_mapped_species1_sorted.bam`). 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

3.2.12 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.

```
cd ~/mapping/
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
```

3.2.13 14. Visualize new mapping

Copy the new data files to your local computer using scp like in step 11. 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 `toy_dataset_mapped_species2_sorted.bam`. You should be able to see them side by side.

Pause here. Check in with your group. Together, discuss and answer these questions for this week's postlab assignment.

Check for understanding:

Q1. 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?

Q2. Do you see evidence of misassemblies or deletions in the reference? What does that evidence look like?

3.3 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 *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, discuss the following question with your group for this week's postlab questions:

Check for understanding:

Q3. 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?

3.3.1 15. Map to your project datasets

Change directory into your project dataset directory folder.

```
cd ~/project_directory
```

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 reads in the dataset `ERR599166_1mill_sample.fasta` against an assembly called `ERR599166_assembly_reformatted.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 CPU so that this goes faster. You'll each use 4 CPUs 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_reformatted.fa ERR599166_assembly_reformatted.btindex
bowtie2 -x ERR599166_assembly_reformatted.btindex -f -U ERR599166_sample.fasta -S␣
↪ERR599166_mapped.sam -p 4
samtools view -bS ERR599166_mapped.sam > ERR599166_mapped.bam
samtools sort ERR599166_mapped.bam -o ERR599166_mapped_sorted.bam
samtools faidx ERR599166_assembly_reformatted.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
```

3.3.2 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.

3.3.3 17. Check for understanding

With your table, discuss and answer the following for this week's postlab:

Q4. Do you see evidence of single nucleotide variants? Biologically speaking, what does this indicate? (Keep in mind that you have mapped metagenomic reads from a whole microbial community against a consensus assembly– this is not reads from an individual vs an individual's reference assembly.)

3.3.4 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_09222020.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.

3.3.5 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. You might have to provide the path to your bed file, or copy it into your project directory.

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

For example:

```
samtools bedcov PROKKA_09222020.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 (replace with the name of your PROKKA file):

```
less PROKKA_09222020.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
```

(continues on next page)

(continued from previous page)

```
##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`.

The `samtools bedcov` command will give you a file that ends in `ORF_coverage.txt`.

3.3.6 19. Matching the Prokka annotations with your ORF coverage

Now you have the coverage for all of your ORFs. To make it easier to read and work with the results, let's match the coverage of each ORF with the Prokka annotation of that ORF so you can more easily read and understand the results. To do that, run this script, substituting in the names and paths of your own ORF coverage file and your faa file from Prokka:

```
python /Accounts/Genomics_Bioinformatics_shared/python_scripts/merge_name.py [ORF_
↪coverage.txt file] [ORFs.faa file]
```

So, for example:

```
python /Accounts/Genomics_Bioinformatics_shared/python_scripts/merge_name.py␣
↪ERR598966_ORF_coverage.txt prokka_project/PROKKA_01252021.faa
```

Your output will be a txt file that matches the name of your ORF coverage file and ends in `_matched.txt`.

3.3.7 20. Calculate coverage in Excel and examine results

Use `scp` to move the new `_matched.txt.` file to your local computer, then open with Excel. The output file should give the name of your contig, the start coordinate, the stop coordinate, the name of your open reading frame, the sum of the per-base coverage, and then the Prokka annotation for that protein. To get the *average* coverage for each ORF, divide the sum of the per-base coverage by the difference between the stop and start coordinates. In other words, type "average coverage" in the top of column H, then one row down, type this and then fill down to the bottom: =F2/(D2-C2)

Pro tip: Rather than drag for 16,000 rows in Excel, highlight the top cell, then scroll down to the bottom of the column, then hold 'Shift' while you click the bottom cell of the column where the data ends, then click Edit –> Fill–> Down.

Now you have the average coverage of every ORF for this particular bam file, and you should be able to see what the annotation of each ORF is. You can explore your results and see what kinds of genes tend to have the highest and lowest coverage by sorting your spreadsheet in different ways.

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).

3.3.8 21. Check for understanding

Take a look at the spreadsheet and discuss with your lab group:

Q5. As you scroll through the data file reporting the average coverage of all of your ORFs, which ORFs had the highest coverage? What do they encode? Speculate on why those genes 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.

Another script that may be useful for your postlab or your final project, but isn't required for this lab:

Let's say you ran a BLAST to identify ORFs of interest, and you want to only get the coverages of the genes that had a match in your BLAST results. To do that, run this script:

```
get_ORF_covg_from_BLAST_hits.py [BLAST file] [ORF coverage file]
```

3.3.9 22. 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. Before doing that, you might want to change the names of some of your files so they are uniform and recognizable (see below example, and substitute your file names for the ones below).

For example:

```
mv PROKKA_09222020.bed ERR598995_assembly_ORFs.bed
```

Then:

```
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
```

3.3.10 23. 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.

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

CHAPTER 4

Week 6: Classifying taxonomy of short reads with mothur

4.1 Introduction

Today we're going to learn how to use sequence data to assess diversity across samples by focusing on just the 16S rRNA gene. You'll recall that 16S rRNA is like a "barcode" gene that we can use to compare and classify who is there. It's often used in microbiome studies. Today we'll learn how to:

- classify the distribution of different types (taxa) of microbes in different datasets (i.e. what % are *Procholoroccus*, 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).

4.2 Using mothur to profile the taxonomy of your dataset

4.2.1 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
```

4.2.2 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 location, 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.

4.2.3 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
```

4.2.4 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)
```

Hot tip #2! You can use the command system() if you want to use Unix commands while you are using mothur. If you can't remember the names of the 16S files you just copied, you can see them by typing this:

```
system(ls)
```

4.2.5 5. Look at the groups file

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

```
system(less groups)
```

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.

4.2.6 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. (In class, we'll talk about why we would ideally use something like operational taxonomic units, or OTUs, to do this analysis. Unfortunately, the Tara metagenomic data is too messy to be able to make nice OTUs, so we're going to classify every sequence.)

```
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.) You have seen this dataset before– it's the one we worked on with Lin!

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.

Part of your postlab assignment will be to explore this dataset and ask a scientific question about it.

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. Check for understanding

Check in with your lab group. Help each other catch up, and then discuss the following check for understanding questions and write down your responses for your postlab:

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.

12. Share data to Google Sheet

Last, please enter the richness and Shannon-Weiner Index data for your sample site in the Google Sheet here . We will use this data on Friday with Lin, and you may use it for your postlab this week.

13. Mini research question

Design a mini research question using the data you've generated today. Remember that you have metadata (i.e. temp, chlorophyll, etc.) available as well in the Google Sheet. Generate a plot or set of plots addressing your research question. Each plot should have a figure caption. (On Friday, we will work on data visualization on these types of datasets with Lin.) Then write a couple paragraphs describing your results, as if this were a mini Results and Discussion section. What kinds of trends do you see? What were you expecting to see, and do your results support your initial expectations? Explain your results in light of what we have discussed in class (and perhaps based on what you have seen in your previous postlabs) about trends in taxonomy and/or diversity in various ocean basins and/or 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):

- Table 1
- Responses to 2 "Check for understanding" questions
- Mini research question

CHAPTER 5

Week 7: 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 contigs 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). We will talk more about this in class.

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 Getting your contigs ready for anvi'o

5.2.1 1. ssh tunnel

As always, boot onto the Mac OS and open up your Terminal application. But don't ssh the normal way! 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.). I'll put that on the white board. Substitute your assigned port number for 8080 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 the co-assembly

In previous weeks, you've been working with assembled contigs of your own samples. This week, we'll be working with a **co-assembly** of all of the Tara samples assembled together. This tends to produce better bins.

Not only that, I've done some Martha Stewart-style pre-baking for you and I have made a *contigs database* out of this co-assembly, ready for use by anvi'o. Basically, I put the co-assembled contigs as well as information from Prokka about ORFs and annotations all into a single database. It is conveniently called `contigs.db`. (If you want more information about how I did this, visit the anvi'o tutorial I linked above.)

To copy the co-assembly database into your own folder, do this:

```
cp /workspace/data/Genomics_Bioinformatics_shared/anvio_stuff/contigs.db .
```

5.2.4 4. Copying the BAM files

Remember that bins are made from information based on **tetranucleotide frequencies** and **coverage**. The contigs database I made for you contains information about tetranucleotide frequencies. To get coverage information, we need to use BAM files based on mappings of reads to the co-assembly. Fortunately for you, I have already mapped all of the Tara metagenomes against the co-assembled contigs. I have also created "Profiles" of those BAM files, which formats them in a way used by anvi'o. (Again, if you want more information about how I did this, see the anvi'o tutorial above.)

They are in this folder:

`/workspace/data/Genomics_Bioinformatics_shared/anvio_stuff/mapped_files`

Copy **up to six** anvi'o profiles from that folder, depending on which samples you're interested in. You have to copy recursively (`cp -r`) because they're folders, not files.

For example:

```
cp -r /workspace/data/Genomics_Bioinformatics_shared/anvio_stuff/mapped_files/mega_
↪assembly_minlength2500_vs_ERR599104_sorted.bam-ANVIO_PROFILE/ .
cp -r /workspace/data/Genomics_Bioinformatics_shared/anvio_stuff/mapped_files/mega_
↪assembly_minlength2500_vs_ERR599090_sorted.bam-ANVIO_PROFILE/ .
cp -r /workspace/data/Genomics_Bioinformatics_shared/anvio_stuff/mapped_files/mega_
↪assembly_minlength2500_vs_ERR599008_sorted.bam-ANVIO_PROFILE/ .
```

5.2.5 5. Merge them together with anvi-merge

Now we have to merge all of these profiles together using a program called `anvi-merge`. See below.

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.'

If you have a sample with tons of contigs, this command may decide not to cluster your contigs together. We're going to force it to do that with the `--enforce-hierarchical-clustering` flag.

This step will take a couple minutes.

```
anvi-merge */PROFILE.db -o SAMPLES-MERGED -c contigs.db --enforce-hierarchical-
↪clustering
```

5.3 Visualizing and making your bins

5.3.1 6. 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
```

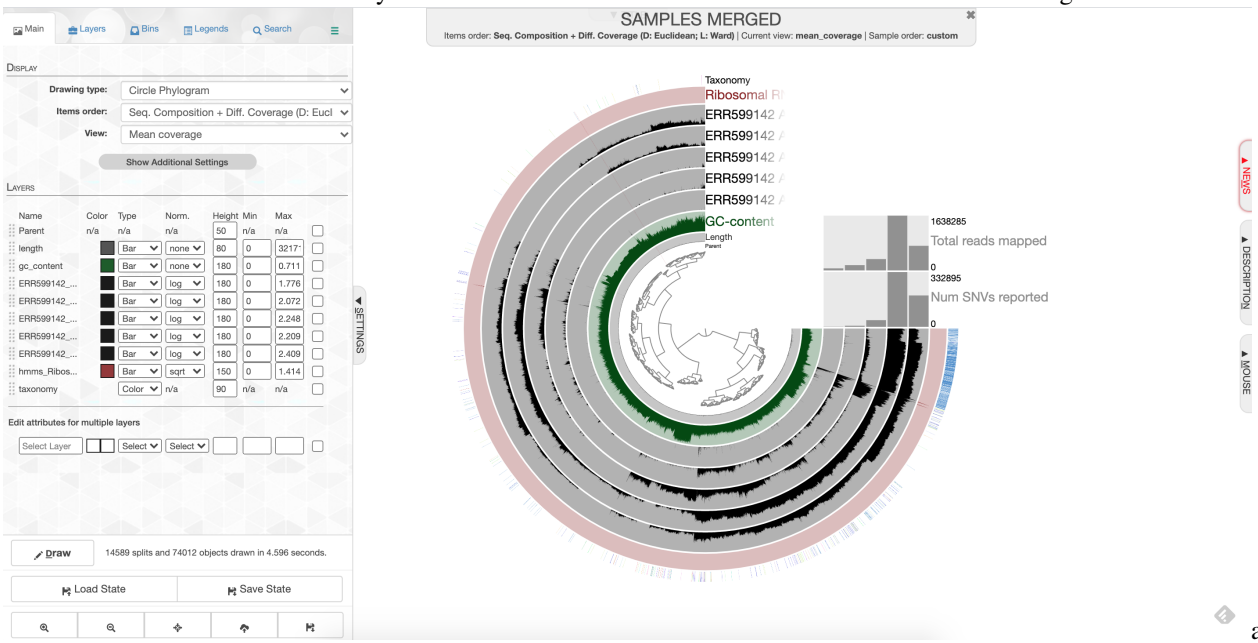
5.3.2 7. Visualize in browser

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. Use the port number (i.e. 8080, 8081, 8082, etc) that 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.

5.3.3 8. 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.

5.3.4 9. 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`)

5.3.5 10. Estimating the metabolism of these genomes

One of the most powerful things about bins is that we can look inside these genomes, see their taxonomy (i.e. who they are), and try to guess what metabolisms they have (i.e. what functions are they performing in the habitat). anvi'o can help us use our Prokka annotations to estimate the metabolism of your bins. We will use `anvi-estimate-metabolism` to do this.

Type this:

```
anvi-estimate-metabolism -c contigs.db -p SAMPLES-MERGED/PROFILE.db -C my_bins --kegg-
↪data-dir /workspace/data/Space_Hogs_shared_workspace/databases/anvio_kegg_database
```

- `-c` lists your contigs database, which has the Prokka information to identify metabolic genes
- `-p` lists your BAM files in the PROFILE folders, which can tell you the coverage of these metabolic genes
- `-C` gives information about the bins you choose
- `--kegg-data-dir` gives directions to the whole KEGG database, which I downloaded earlier for you, to help categorize these genes using the KEGG database

5.4 Analyzing your bins

All right! You now have an output file called `kegg-metabolism-modules.txt`. I recommend that you use `scp` to download this to your computer, and take a look at it in Excel.

Here's what the columns mean:

- `bin_name`: the bin name that anvi'o assigned your bins as you were making them, i.e. "Bin_1."
- `kegg_module`: the pathway that the gene can be found in, as assigned by KEGG. See here for more information.
- `module_name`: the name of the KEGG module in which the gene is found
- `module_class`: the type of module it's found in. It can be a "pathway module," or a gene in a metabolic pathway, a "signature module", or a gene that characterizes a specific phenotype like drug resistance or pathogenicity, or a "reaction module," a set of genes that catalyze successive reactions (these are usually extensions of specific metabolic pathways.)
- `module category`: is the broadest category in which the metabolic pathway is found, i.e. "Carbohydrate metabolism"
- `module subcategory`: one level down in terms of categories in which the metabolic pathway is found, i.e. "Carbon fixation"
- `module definition`: the list of genes that are found in a pathway or module, listed by their KO (or "KEGG Orthology") number, which is how KEGG labels different genes
- `module completion`: how complete the pathway is. If your metabolic pathway has 10 genes in it, and 6 of them are present in your bin, this value will be 0.6.
- `module_is_complete`: this will only say 'TRUE' if your module completion value is 75% or above.
- `kofam_hits_in_module`: tells you exactly which genes from the pathway were present in your bin
- `gene_caller_ids_in_module`: tells you which gene numbers from Prokka (and re-numbered by anvi'o) were in your pathway.

5.5 Postlab assignment

For this week's postlab assignment, there are a few "check for understanding" questions and a mini research question.

5.5.1 A. Check for understanding

1. In the anvi'o visualization, what does each of the "leaves" of the tree represent? What is the clustering meant to indicate (i.e. what do the branches on the tree represent)?
2. In the anvi'o visualization, what does each of the grey rings represent? Which file contained the data needed to create each of those rings?
3. Everyone in the class was working with the same co-assembly, and yet the clustering of the contigs in the anvi'o wheel might have looked different between different people in the class. Using what you know about how bins are made, explain why this might be the case.

5.5.2 B. Mini research question

Use your anvi'o outputs to do some data exploration to ask and answer a scientific question. I'd recommend paying the most attention to two output files: `kegg-metabolism-modules.txt` and `SAMPLES-MERGED/SUMMARY_my_bins/bins_across_samples/mean_coverage.txt`, but you're welcome to use any of the anvi'o output you like.

Turn in Figure 1 with a figure caption, the three "check for understanding" questions, and your mini research question in by lab next week on the class Moodle page.

CHAPTER 6

General Stuff

6.1 ssh-ing into baross

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

6.2 Using screen/ tmux

(We have been using screen in class.)

See: Cheat Sheet

^ = control key

Action	tmux	screen
start new session	`tmux`	`screen -S screen_name`
detach from current session	`^b d`	`^a ^d`
kill current session		`^a ^k`
re-attach detached session	`tmux attach`	`screen -r screen_name`
list sessions	`^b s`	`screen -ls`

6.3 File Transfer

6.3.1 Filezilla

- Open Filezilla
- Host: sftp://baross.its.carleton.edu
- Username: Carleton username

- Password: Carleton password
- Port: 22
- Click QuickConnect

6.3.2 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]
```

6.3.3 Open via FTP with BBEdit

- In BBEdit, go to File –> Open from FTP/SFTP Server. . .
- Server: baross.its.carleton.edu
- Check the SFTP box
- Port: 22
- User: Carleton username
- Password: Carleton password
- Path: /Accounts/Carleton username
- Click Connect

CHAPTER 7

Bioinformatics Tools

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

7.1 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.

7.2 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]
```

7.3 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.

7.3.1 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.

7.3.2 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.

7.4 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]
raxmlHPC-PTHREADS-AVX -f a -# 20 -m PROTGAMMAAUTO -p 12345 -x 12345 -s [example_
↪sequence_file_aligned.afa] -n [example_tree_name.tree] -T 4
```

Open the RAxML_bipartitions.example_tree_name.tree file in ITOL.

7.5 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]
```

7.6 mothur

Analyze 16S data to find taxonomy, OTUs

The protocol for week 6 laid out mothur step by step. The mothur wiki is another good source of information.

7.7 anvi'o

Make bins from metagenomic assemblies and BAM files

The protocol for week 7 laid out anvi'o step by step. The anvi'o tutorial is another good source of information.

CHAPTER 8

Legacy Items

8.1 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.

8.2 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 9

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!

9.1 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.)

9.1.1 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.

9.2 Step 2: Sphinx

9.2.1 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.

9.2.2 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.

9.3 Step 3: Github

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

9.3.1 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
```

9.3.2 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 10

Authors

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