Carleton Bioinformatics Documentation

Rika Anderson

May 09, 2018

Protocols:

CHAPTER 1

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

Rika Anderson, Carleton College, Dec 23, 2017

1.1 Connecting to Liverpool

1.1.1 1. About Liverpool

We are going to do most of our computational work on a remote server (or computer) called Liverpool, which is a Linux virtual machine set up by Carleton ITS for our course. Liverpool basically lives in a computer somewhere in the CMC. You'll be accessing it from the lab computers, but you could also access it from your own computer at home. First, you have to learn how to access Liverpool.

Boot as a Mac user on the lab computer.

1.1.2 2. Opening Terminal

Find and open the Terminal application (it should be in "Applications" in the folder called "Utilities").

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 our class server, liverpool). Type the following:

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

1.1.4 4. ssh (cont.)

You should see something like this: `[your username]@liverpool.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 liverpool, you end up in your own home directory. Each of you has your own home directory on liverpool. 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 liverpool, also known as your **path**. But before we actually run any processes, we need to learn a few important things about using a remote shared server.

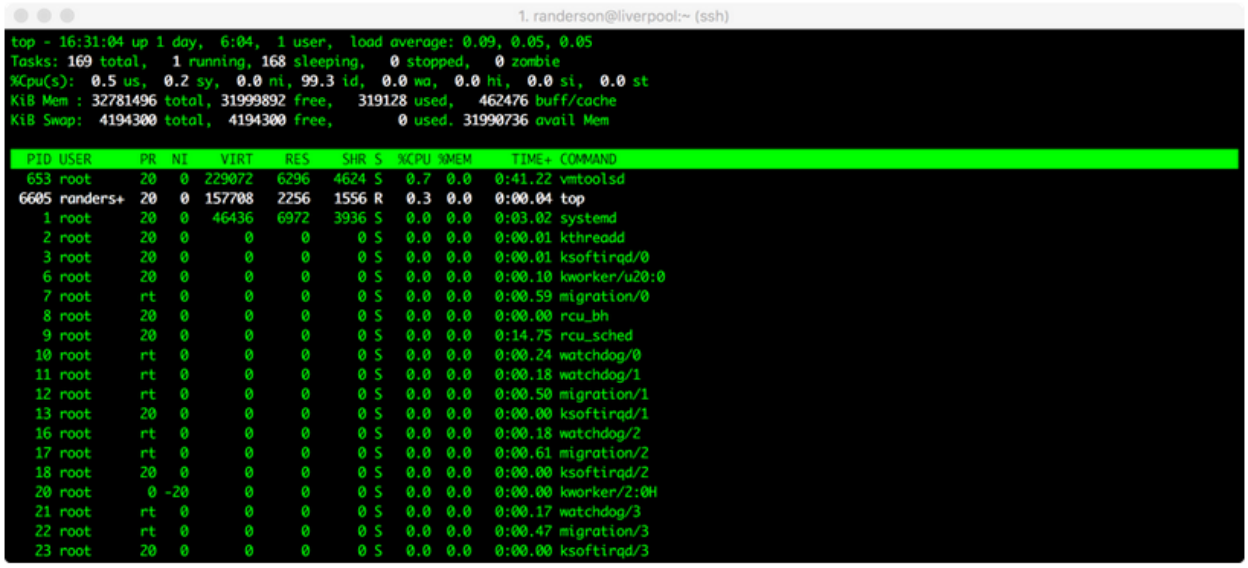
Important things to know about when using a remote shared server

1.1.7 7. top

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

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

You should see something like this:



Here, we can see that randerson is the only user on liverpool, 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. **Liverpool has 10 CPUs and 32 gigabytes of RAM total.** Therefore, we can use up to 10 CPUs. 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. If it looks like all 10 CPUs are already being used at a particular time, you might want to wait until later to run your own process.

To quit out of top, type: `q`

1.1.8 8. screen

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

Your terminal will open a clean window. You are now in a screen session. 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`. (Now you've re-entered your screen session.)

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.

1.2 Creating an assembly and evaluating it (toy dataset)

1.2.1 9. mkdir

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

Make a new directory in your home folder:

```
mkdir toy_dataset_directory
```

1.2.2 10. cd

Change directory into your toy dataset directory:

```
cd toy_dataset_directory
```

1.2.3 11. Copy toy dataset

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

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

1.2.4 12. Run idba-ud on toy dataset

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

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

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

1.2.5 13. cd to output directory

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

```
cd toy_assembly
ls
```

1.2.6 14. The output files

You should see several files, including these:

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

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

1.2.7 15. Examine output

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

```
less scaffold.fa
```

1.2.8 16. Fasta file

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

1.2.9 17. Evaluate assembly quality

Now we're going to evaluate the quality of our assembly. One measure of assembly quality is N50, which is the contig length for which the collection of all contigs of that length or longer contains at least 50% of the sum of the lengths of all contigs. We could theoretically calculate the N50 by hand, but often times you'll get many, many contigs and it's very tedious to do by hand. So we will use an assembly evaluator called Quast. Run it on your toy dataset. (If you've been copying and pasting, it may not work here; you may have to type it in.) Here is what the commands mean:

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

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

1.2.10 18. View output with x2go

Quast gives you some nice visual outputs that are easiest to see if you log in to x2go. More detailed instructions are here:

https://wiki.carleton.edu/display/carl/How+to+Install+and+Configure+x2go+client

1. Find x2go on your lab computer and open it.
2. Click "New session" in the top left of the screen (it's an icon with a piece of paper with a yellow star on it.)
3. Enter "liverpool.its.carleton.edu" in the Host box. Choose the SSH port to be 22. The session type should be XFCE. You can set the session name to "Liverpool" or whatever you like. Click OK.
4. Click on your newly created session, which should appear on the right side of the window.
5. Log in with your Carleton account name and password.
6. Navigate to the directory called "toy_assembly_quast_evaluation."
7. Double-clock on the file called "report.html" and take a look. It reports the number of contigs, the contig lengths, the N50, and other statistics for each of the assembly files you entered. The graph at the bottom shows the cumulative length of your assembly (y-axis) as you add up each contig in your assembly (x-axis). Many of your lab questions for this week are based on the quast assembly report.
8. When you are done with x2go, be sure to click on "Log Out" when you log out!

1.2.11 19. Bookkeeping

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

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

1.3 Create and evaluate assembly (project dataset)

1.3.1 20. screen

You're now going to run an assembly on your project dataset. Since this will take about 10-20 minutes, I recommend running this on screen. Type:

```
screen
```

1.3.2 21. cd

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
```

1.3.3 22. Copy project dataset

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

1.3.4 23. Start assembly

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

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

1.3.5 24. Detach from screen

This is likely to take a little while. After you've started the process, detach from screen:

```
Ctrl+A d
```

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

When IDBA-UD finishes, it will say:

```
invalid insert distance
Segmentation fault
```

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

1.3.6 25. Rename assembly

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

This way we will have standardized names and save ourselves future heartache. (For a reminder on how to change file names, see your Unix cheat sheet.)

1.3.7 26. Run anvi

Run `anvi-script-reformat-fasta` on your completed project assemblies (see step 19 for instructions).

1.3.8 27. Exit

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

```
exit
```

NOTE!!! If you type this while you are still in screen it will kill your screen session! Be sure to detach from screen first if you want to let your process keep running after you log off the server.

These assemblies will likely take some time. Take advantage of your time in lab to discuss the following questions with your colleagues. I encourage collaboration and sharing of ideas, but I expect each of you to submit your own assignment, which should reflect your own ideas and understanding of the responses.

1.4 Questions to answer for this week

1.4.1 28. Questions

Consider and answer the following questions and submit a Word document with your responses on the Moodle by lab time next week.

Check for understanding:

Q1. (a-d) About your toy dataset assembly:

- How many reads did you start with in the raw data file for your toy dataset, prior to assembly? (Hint: refer to the Unix tutorial you did, and use grep to search for the '>' symbol that denotes each new sequence. The > symbol also means something in Unix, so be sure to use quotes around it!)
- 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').
- Examine the Quast output. How does the N50 of your scaffold file compare to your contig-100 file? Explain why.
- Examine the Quast output. Which assembly output file had the most contigs? The fewest? Explain why.

Critical thinking:

Q2. Let's say that in preparing your samples for sequencing, you sheared your DNA to create an insert size (DNA sequence length) of 200 bp, and you sequenced them on an Illumina machine with 200 bp reads. How would you expect your scaffolds file to differ from your contig-100 file? What if you had instead selected an insert size of 500 bp? Explain your reasoning.

Q3. Describe why a CRISPR array (see here for a reminder of what that looks like) might present a problem for assembly and what strategies you might use to overcome this problem.

Q4. If you were testing different types of assembly software on a metagenome with high coverage, and one used k-mers of length 40 and the other used k-mers of length 20, how would you expect the N50 and total assembly length to differ and why? In contrast, if you were testing them on a metagenome with low coverage, would you expect the same results?

Q5. Last week, we sampled water bodies within the Cannon River watershed. In the winter, you tend to get a very diverse microbial community, with lots of single nucleotide variants differentiating the microbes. In contrast, in spring or fall after a heavy rainfall, you might expect that nutrient runoff would go into the lakes, leading to a bloom of one specific type of microbe with very few single nucleotide variants, creating a less diverse microbial community. How might you expect your assembly N50 to differ between the winter and spring/fall communities? (You should include a discussion of coverage and de Bruijn graph construction in your answer.)

CHAPTER 2

Week 4: Calling open reading frames with Prodigal, using BLAST, annotating genes with Interproscan

Rika Anderson, Carleton College

2.1 Pre-lab notes

2.1.1 1. Start a notebook

Before we start this week, I want to encourage you all to start a computational lab notebook. This can take whatever form you want– I often will just open a text document on my computer and save it on the cloud somewhere– but it will be crucial that you keep a record of all the commands you run, especially for your project datasets. That way, if you get a funny result, or if you want to repeat an analysis, you can always go back to your lab notebook to see exactly what command you ran. I will sometimes include notes to myself about the analysis, or observations of the graphs I make. These notebooks are just for you– I won't be checking them– but now that we're about to get into the real analysis, I strongly recommend that you start one for your project datasets and maintain them throughout the rest of this course.

2.2 Logging in to the remote server

2.2.1 2. Login

Boot as a Mac on the lab computer.

2.2.2 3. Terminal

Find and open the Terminal application (`Applications/Utilities`).

2.2.3 4. Connect to baross

This week, we're going to be using a different server called baross, which is my research server. This is because liverpool is actually a pretty small server– it only has 10 processors. In contrast, baross has 96. Liverpool and baross share the same storage space, so all of your files on liverpool are mirrored on baross. This means that you should see the same file structure regardless of which server you log into, but the amount of computational power available to you will be different depending on which server you logged into. Please note: this is Rika's lab's research server, so please do NOT use this server unless instructed to.

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

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

2.3 Finishing up from last week

2.3.1 5. Copy assembly into class folder

Your assemblies from last week should have finished by now. If you haven't already, please use the `anvi-script-reformat-fasta` script on your assembled project datasets to make sure they're properly formatted and ready to go.

When that is done, please copy your newly formatted, assembled project assemblies and put them into the folder that we'll be sharing as a class, substituting the placeholder below with the name of your assembled file. This will be the shared folder for all of the data that you generate as a class.

```
cp [name of your formatted, assembled file] /usr/local/data/class_shared
```

2.4 Searching for open reading frames

2.4.1 6. cd to toy dataset directory

First, we're going to learn how to identify open reading frames (ORFs) on your assembled toy dataset from last week. To do that, we'll use a program called Prodigal. There are lots of programs that can identify ORFs, and there are strong distinctions between ORF callers for eukaryotes versus bacteria/archaea/viruses. Prodigal is one of the best ORF callers for microbial genomes and it is free, so that's what we're using today. If you find yourself wanting to identify ORFs on some sequence in the future, most ORF callers should work in a similar manner to Prodigal.

Go to your toy dataset directory:

```
cd ~/toy_dataset_directory
```

2.4.2 7. Copy toy dataset

We're going to try out Prodigal on an assembled toy dataset to learn how this works. (Even though you assembled this toy dataset last week, we're going to use a subsample of that assembled toy dataset in order to be more computationally efficient.) Copy `toy_dataset_assembled_subsample.fa` into your `toy_assembly` directory:

```
cp /usr/local/data/toy_datasets/toy_dataset_assembly_subsample.fa toy_assembly
```

2.4.3 8. mkdir

Now that you're in the toy dataset directory, make a new directory and change into it.

```
mkdir ORF_finding
cd ORF_finding
```

2.4.4 9. Run prodigal

Now run Prodigal on your toy assembly subsample:

- 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.

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

2.4.5 10. View output

You should see two files, one called `toy_assembly_ORFs.gbk` and one called `toy_assembly_ORFs.faa`. Use the program `less` to look at `toy_assembly_ORFs.faa`. 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.

```
less toy_assembly_ORFs.faa
```

2.5 Finding the function of our open reading frames/genes/proteins

2.5.1 11. Copy sequence of first ORF

Now we have all of these nice open reading frames, but we don't know what their function is. So we have to compare them to gene databases. There are lots of them out there. First, we will use BLAST and compare 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.

Take a look at your Prodigal file again using the program `less` (see above). Copy the sequence of the first ORF in your file. You can include the first line that includes the > symbol.

2.5.2 12. Navigate to NCBI site

Navigate your browser to the BLAST suite at: https://blast.ncbi.nlm.nih.gov/Blast.cgi. Select Protein BLAST (blastp).

2.5.3 13. Paste sequence

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

2.5.4 14. Blast off

While that's running, go back 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 sequnces, which can include whole genomes.

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

2.5.5 15. Post-lab questions

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

Check for understanding:

Q1. For both your `blastp` and `tblastn` results, list:

- The top hit for each of your database searches
- The e-value and percent identity of the top hit
- What kind of protein it matched
- What kind of organism it comes from

Q2. Take a look at the list of hits below the top hit.

- Do they have the same function?
- Do think you can confidently state what type of protein this gene encodes? Explain.
- Describe the difference in your `tblastn` and `blastp` results, and explain in what scenarios you might choose to use one over the other.

2.5.6 16. Introduce Interproscan; activate screen

This works well for a few proteins, but it would be exhausting to do it for every single one of the thousands of proteins you will likely have in your dataset. We are going to use software called Interproscan to more efficiently and effectively annotate your proteins. It 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.

Interproscan takes a long time to run. With a big dataset, it could run for hours. So let's open a screen session. (Note: for your toy dataset, this will take minutes, not hours, so screen isn't entirely necessary. But it's good to practice– and for your project datasets, you will definitely want to be in screen.)

```
screen
```

2.5.7 17. sed

Interproscan doesn't like all of the asterisks that Prodigal adds to the ends of its proteins. So get rid of them by typing this:

Explanation: `sed` is another Unix command. It is short for "stream editor" and it is handy for editing files on occasion, like when you're trying to do a "find/replace" operation like you might do in Word. Here, we're using sed to replace all of the asterisks with nothing.

```
sed 's/\*//g' toy_assembly_ORFs.faa > toy_assembly_ORFs.noasterisks.faa
```

2.5.8 18. Run interproscan

Now run interproscan.

- 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.

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

2.5.9 19. Terminate screen

When it's done, terminate your screen session.

(You could also try `Ctrl + A + k` – Hit "Control" and "a" at the same time, and then release and immediately press "k".)

```
exit
```

2.5.10 20. FileZilla

Now let's take a look at the results. The output should be called `toy_assembly_ORFs.noasterisks.faa.tsv`. The output is a text file that's most easily visualized in a spreadsheet application like Excel. The easiest way to do that is to copy this file from the server to your own desktop and look at it with Excel. We're going to learn how to use FileZilla to do that. (Note, yes, you're connecting FileZilla to liverpool. Remember that baross and liverpool share the same file structure, so it shouldn't matter which one you connect to.)

Locate FileZilla in the Applications folder on your computer and open it. Enter the following:

1. For the Host: sftp://liverpool.its.carleton.edu
2. For the Username: your Carleton username
3. For the password: your Carleton password
4. For the Port: 22
5. Then click QuickConnect.

The left side shows files in the local computer you're using. The right side shows files on liverpool. On the left side, navigate to whatever location you want your files to go to on the local computer. On the right side, navigate to the directory where you put your Interproscan results. (`/Accounts/yourusername/toy_dataset_directory/ORF_finding/toy_assembly_ORFs.noasterisks.faa.tsv`) All you have to do is double-click on the file you want to transfer, and over it goes!

2.5.11 21. Open tsv file

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

1. Protein Accession (e.g. `P51587`)
2. Sequence MD5 digest (e.g. `14086411a2cdf1c4cba63020e1622579`)
3. Sequence Length (e.g. 3418)
4. Analysis (e.g. database name– Pfam / PRINTS / Gene3D)
5. Signature Accession Number (e.g. PF09103 / G3DSA:2.40.50.140)
6. Signature Description (e.g. BRCA2 repeat profile)
7. Start location
8. Stop location
9. Score - is the e-value of the match reported by member database method (e.g. 3.1E-52)
10. Status - is the status of the match (T: true)
11. Date - is the date of the run

2.5.12 22. Post-lab questions

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

Check for understanding:

Q4. Take a look at the annotation of the contig you BLASTed earlier. If there is more than one hit, that's because Interproscan is returning hits from multiple protein databases. What is the description of the hit with the best score, according to Interproscan? (Remember, Google can be a handy resource sometimes.) Does that match your BLAST results?

Searching for matches within your own database

2.5.13 23. pwd

We just learned how to compare all of the proteins within your dataset against publicly available databases using Interproscan. This will be handy later on, because it gives you a nice list of all of the proteins in your assembly. But you may also wish to choose a specific protein of interest and see if you can find a match within your dataset. Why? There are a few reasons why you might do this. For example, you may have a gene you're interested in that's not available in public databases, or is relatively unknown. It won't show up in the Interproscan results. Or maybe you want to know if any of the ORFs or contigs in your dataset have a match to a particular sequence, even if it isn't the top hit that Interproscan found.

We will learn how to use BLAST to search for a gene, using your own dataset as the databse to search against.

First, make sure you are in the correct directory.

```
pwd
/Accounts/[your username]/toy_dataset_directory/ORF_finding
```

2.5.14 24. 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 toy_assembly_ORFs.faa -dbtype prot
```

2.5.15 25. Find protein

Now your proteins 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:

http://pfam.xfam.org/family/PF03787.

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.

2.5.16 26. Find protein (cont.)

Click on "1319 sequences" near the top.

2.5.17 27. 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."

2.5.18 28. Transfer file

Use FileZilla to transfer this file to your `ORF_finding` directory on baross.

2.5.19 29. View file

Take a look at the Pfam file using less or using a text editing tool on the local computer. It is a FASTA file with a bunch of "seed" sequences that represent a specific protein family from different organisms. If you want to learn more about a specific sequence, you can take the first part of the fasta title (i.e. "Q2RY06_RHORT") and go to this website: http://www.uniprot.org/uploadlists/ and paste it in the box. Then select from "UniProt KB AC/ID" to "UniProtKB" in the drop-down menu on the bottom. You will get a table with information about your protein and which organism it comes from (in this case, a type of bacterium called Rhodospirillum).

2.5.20 30. BLAST it

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

- `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 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.

```
blastp -query PF03787_seed.txt -db toy_assembly_ORFs.faa -evalue 1e-05 -outfmt 6 -out␣
↪PF03787_vs_prodigal_ORFs_toy.blastp
```

2.5.21 31. 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 FileZilla and open it in Excel.)

2.5.22 32. 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

2.5.23 33. Post-lab questions

Take a look at your BLAST results and answer the following questions for the Word document you will submit on the Moodle.

Check for understanding:

Q5. (a-e)

- Which protein among your Pfam query sequences had the best hit?
- What was the percent identity?
- What organism does the matching Pfam protien query sequence come from?
- 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? 34

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

```
blastp -query PF03787_seed.txt -db toy_assembly_ORFs.faa -evalue 1e-02 -outfmt 6 -out␣
↪PF03787_vs_prodigal_ORFs_toy_evalue1e02.blastp
```

2.5.24 35. Post-lab questions

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

Check for understanding:

Q6. How do these BLAST results differ from your previous BLAST? Explain why.

2.5.25 36. Try another gene

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

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

```
nano DNA_pol_T_aquaticus.faa
```

2.5.26 37. BLAST

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

```
blastp -query DNA_pol_T_aquaticus.faa -db toy_assembly_ORFs.faa -outfmt 6 -out DNA_
↪pol_T_aq_vs_prodigal_ORFs_toy.blastp
```

2.5.27 38. Examine results

Take a look at your blastp output using less. Notice the e-value column, second from left. Most of your e-values for this range between 2.4 (at best) and 7.7 (at worst). These e-values are TERRIBLE. This means you probably didn't have very good hits to this protein in your dataset. (What constitutes a "good" e-value is debatable and depends on your application, but usually you want it to be at least 0.001 or lower, as a general rule of thumb.)

2.5.28 39. Post-lab questions

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

Check for understanding:

Q7. Describe the protein you chose and how you found the sequence for that protein.

Q8. Show the command you executed for the blast.

Q9. Where there any matches? If so, which contig was the best match? Applying these analyses to your project datasets

2.5.29 40. Analyze project dataset

Now we're going to apply these analyses to your own datasets for your projects.

Identify open reading frames on your project assembly using Prodigal, then determine their functions using Interproscan. Use this lab handout for reference. **Don't forget to use the 'fixed' version of your contigs** (the version that has been run through `anvi-script-reformat-fasta`.) Your TAs and I are here to help if you have questions or get stuck—usually the problem is a typo or you're in the wrong directory, so check that first! **Your Interproscan runs will take several hours, maybe days. Be sure to run them on screen so they can run overnight. In order to spread out the processes, I will ask that 6 of you start your Interproscan runs on baross, and 5 of you start your runs on liverpool.** And before you start competing for one server over another, it won't matter in the long run– all of your runs will be done by the end of the week. (I think.)

When you are finished, copy your assembly, your ORFs, and your Interproscan results to our class' shared folder. For example, if your Interproscan results are called `ERR598966_assembly_ORFs.noasterisks.faa.tsv`, copy them over like this:

```
cp ERR598966_assembly _ORFs.noasterisks.faa.tsv /usr/local/data/class_shared
```

2.5.30 41. Post-lab assignment

For your post-lab assignment, compile all of the Moodle questions above into a single document. There are nine 'check for understanding' questions listed above.

Critical thinking question:

Finally, develop a question about your project dataset that you can answer using BLAST or using your Interproscan results. Remember that you can search for specific genes in your own dataset, or you can use the NCBI BLAST website to compare your own contigs to the NCBI non-redundant database, or you can search through your Interproscan results. For example, your question could be something like: 'how many different open reading frames in my dataset have a match to photosystem II?' or 'what genes in the NCBI non-redundant database have the best match to my longest contig?' Describe your question, explain how you went about answering it, and describe your results.

You should clearly define a specific biological question related to your dataset. The description of your methods should be thorough (specify important details, like what database you obtained sequences from, or what e-values

you specified). Your description of your results should be detailed and must link back to some sort of biological interpretation.

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

CHAPTER 3

Week 5: Aligning with MUSCLE and Making Trees with RAxML

To be created. . .

CHAPTER 4

Week 6: Mapping with Bowtie2

To be created...

CHAPTER 5

Week 7: Binning Genomes with Anvi'o

To be created...

CHAPTER 6

Week 8: Classifying taxonomy of short reads with mothur

To be created. . .

CHAPTER 7

General Stuff

7.1 ssh-ing into liverpool / baross

Liverpool:

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

Baross

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

7.2 Using screen/ tmux

See: Cheat Sheet

^ = control key

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

7.3 File Transfer

7.3.1 Filezilla

- Open Filezilla

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

7.3.2 Secure Copy

From liverpool to local:

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

From local to liverpool:

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

CHAPTER 8

Bioinformatics Tools

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

8.1 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!)

9.1 Step 1: Markdown

Learning:

- Markdown is a simple way to format text.
- You can indicate in plain-text (ie, by typing special symbols rather than pushing application-specific buttons) that things should be bolded or italicized or seen as headers or links.
- See commonmark.org/help for quick reference. (Though note some Markdown "engines" support other syntax, beyond and even different than the syntax specified by CommonMark.)

Using:

- A nice way to edit markdown and see it rendered into HTML is to use the Atom text editor, with the fantastic extension Markdown Preview Enhanced.

9.2 Step 2: Sphinx

Installing:

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`

Using:

- Edit the file `index.rst` to alter which files are included
- Edit the file `conf.py` to alter settings
- 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.

  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:

```
# see what has changed
git status

# stage modified files
git add -u

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

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

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

CHAPTER 10

Authors

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