Week_3.md - Grip

Welcome to Week 3 of ESPM 112 lab!

This week we're going to be looking at metagenome assembly- what it is, how to do it, and best practices.

Your samples are enormous (some of the uncompressed .fastq files are >65GB!) so we're not going to be able to do metagenome assembly on all of these today.

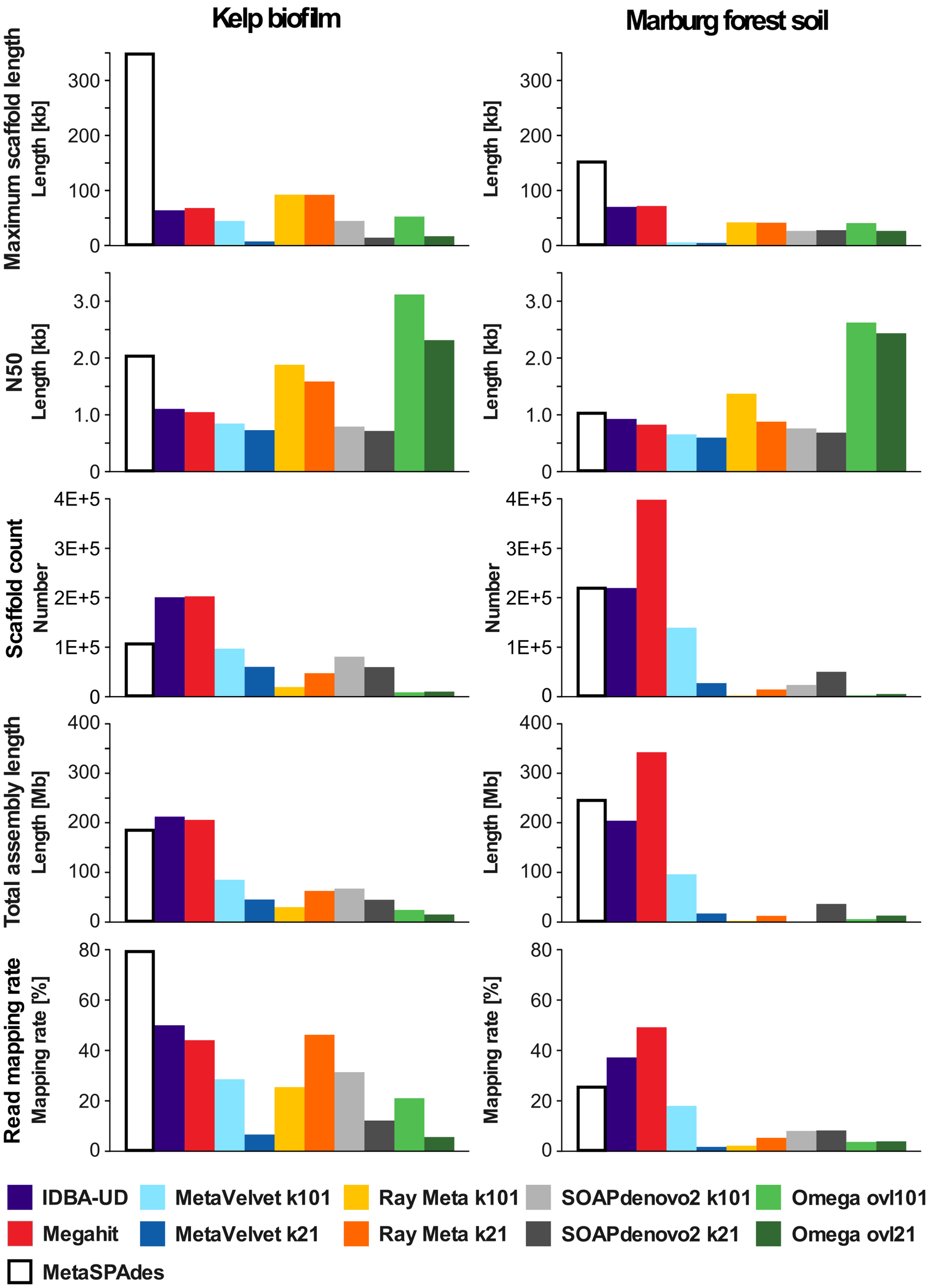
What we are going to do is an overview of metagenome assembly- what it is, how to run it, and what software to use.

First, here's a link to a nice paper by Vollmers et al. that describes the most popular software packages for metagenome assembly. It'll be a nice resource for you in the future if you encounter this again.

Second, let's go over the methods we use in our lab and why we use them.

Our lab uses `idba_ud`, as we tend to get the best results with it (and it has a nice built-in scaffolder). This is by no means a one-size-fits-all solution; different assemblers work to different degrees depending on the type of sample you're working with, its environment of origin, and your sequencing depth.

For a nice example, see the image below from the paper mentioned above:



As you can see, different assemblers win out over others when looking at particular metrics, but none is consistently better than all the others based on all metrics across different sample types. It's up to you to decide which one is best for your particular situation, based on the particular traits of each assembler (which are well described in the Vollmers et al. paper above).

Our lab uses a mixture of `MEGAHIT` and `idba_ud`, so I'll give you the option of running either of those today; both are installed on the class server.

Review: Quality and Trimming

- Last week, as you'll recall, we looked at how to investigate the quality of your reads with FastQC, as well as how to trim your reads based on quality using Sickle.

- Since we subset the reads last time due to computational constraints, I've provided two example FastQC reports for you: One pre-trimming and one post-trimming. They're stored at`/home/jwestrob/fastqc_output/S3_002_pre_trimming_fastqc.html` and `/home/jwestrob/fastqc_output/S3_002_trimmed_fastqc.html`, and they're both also stored in this github repository. Try opening them with your browser and pulling them up side-by-side. What are the main differences you see? How dramatic is the difference in quality? (They shouldn't be terribly different)

This week's material: Assembly and Assembly Statistics

Section 1: Assembly

First, we're going to set up a practice assembly. Navigate to `/class_data/practice_assembly` and take a look at what's there.

You'll notice there's two types of files here: `.fastq` and `.fa`. The `.fa` files are FASTA format, whereas the `.fastq` are in FASTQ format. You'll often see FASTA files with extensions like `.fa`, `.fasta`, `.fna`, and `.faa`. These all mean mostly the same thing, which is that it's in FASTA format. However, two are more specific: `fna` stands for **f**asta **n**ucleic **acid** (DNA FASTA) and `faa` stands for **f**asta **a**mino **a**cid (Protein FASTA).

Now you're going to be practicing genome assembly. You have two options here: assemble your `.fastq` files with `MEGAHIT`, or assemble your `.fa` file with `idba_ud`. Choose one and stick with it! I'll give you help below.

Subsection: Tmux

This is going to take a super long time to run, so you're not gonna want to have one terminal window open the whole time. Allow me to teach you about a nifty program called `tmux`.

`tmux` allows you to make terminal windows that you can leave running (even when you close your connection!) and come back to later. It's really nice. There are other alternatives in bash, such as `nohup` and `screen`, but let's focus on this for now.

- You're going to want to make a new window before you start anything. Try this:
  - `tmux new -s assembly`
- Congrats! Now you're in a new `tmux` window called 'assembly'. To exit this window and leave it running, press `CTRL+b`, then `d`. If you've done that and you want to get back, use:
  - `tmux attach -t assembly`

IMPORTANT: Only one person per group should do the rest of this section!

Subsection: Assembly prep

Now we're going to prepare to run an assembly. Choose your reads, and do the following:

If you're going to assemble .fastq files using MEGAHIT:

- `ln -s /class_data/practice_assembly/*.fastq ~`

If you're going to assemble .fa files using idba_ud:

- `ln -s /class_data/practice_assembly/*.fa ~`

This will create what's called a *symbolic link* in your home directory (~) - it's like copying over a file, but you don't actually make a new copy. You can just see the filename and operate on it as if you had copied it. If you remove this link, the original will be safe and sound in its original directory.

Now, we're going to do actual assembly. Remember, **only one person per group should execute one of the following commands! We only have so many compute resources.**

I also highly encourage you to look into potential alterations to these commands, of which there are many. The commands I provided to you will work, but try using either of the following commands to see more parameters to play with, and please ask me about them in class! (The following commands show the help menus for these two assemblers.)

(don't play with the number of threads though)

```
idba_ud -h
```

```
megahit -h
```

If you're assembling using `idba_ud`:

```
idba_ud --pre_correction --min_contig 500 -r 4_milli_trimmed_raw.fa --num_threads 4 -o 4milli_idba_ud_practice
```

If you're assembling using `megahit`:

```
megahit -1 4milli_trimmed.R1.fastq -2 4milli_trimmed.R2.fastq -t 4 -m 0.13
```

(The `-m 0.13` flag limits the process to 13% of the system's memory, ensuring that we can run at least 7 of these assemblies on the server at once.)

Now these should take about 20 minutes for `idba_ud` and 4 minutes for `MEGAHIT`.

Section 2: Assembly Statistics

- Take a look at the directory for your sample (e.g. /class_data/S3_002_000X1 or similar). *If you don't remember this ask me for help!*
- You'll see two subfolders now - `assembly.d` and `raw.d`. We've already assembled this data, since it's absolutely enormous and would take a really long time to assemble on the class server. You'll find the reads you were working with last week in `raw.d` and the pre-made assemblies in `assembly.d`.
- We're going to do a little bit of post-assembly quality control using the scaffolds now, which is just as important as investigating the quality of the reads pre-assembly.

We'll do this by using `quast.py`- try it with the following commands.

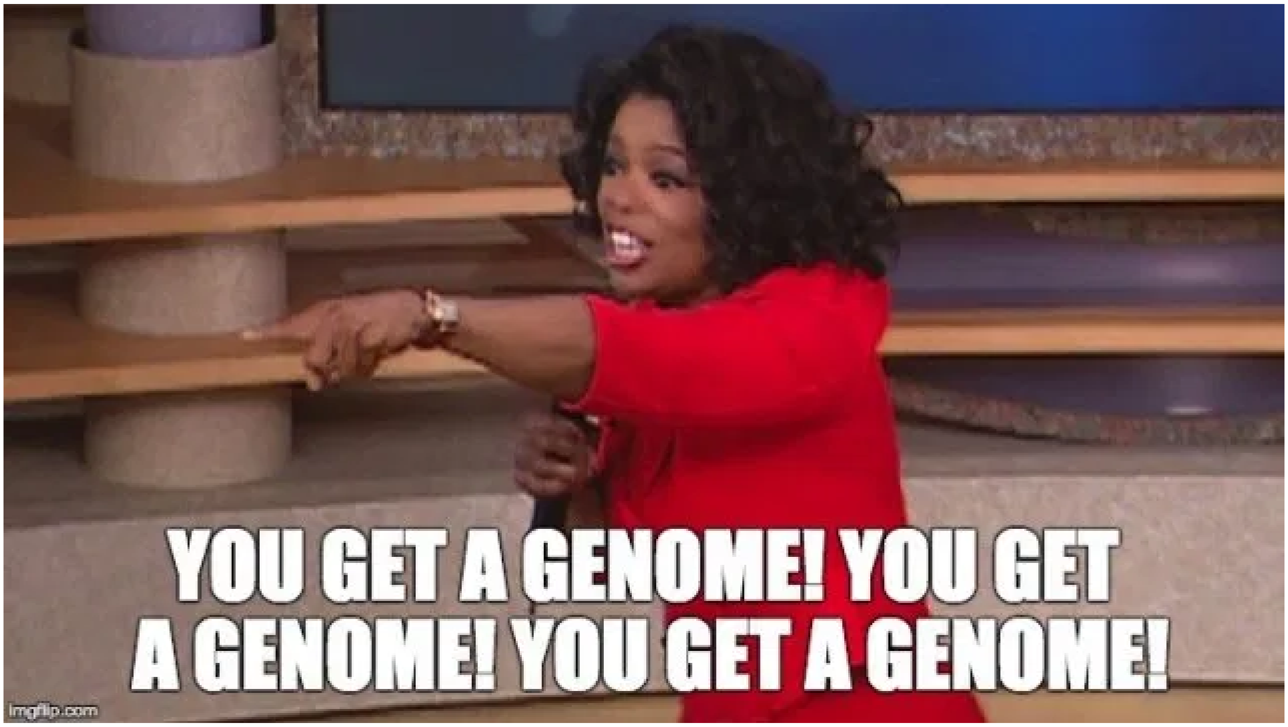
```
mkdir ~/quast_output
/home/jwestrob/quast-5.0.2/quast.py -t 1 [YOUR CONTIG FILENAME HERE] -o ~/quast_output
```

(It will tell you that `python-matplotlib is missing or corrupted`. Don't worry about it.)

Navigate into `quast_output`. Use `realpath report.html` to get the full path of the file, download it to your local machine, and visualize it. What do you see? If you have time, and your assembly is completed, try this on your completed assembly as well- how different are the stats? In which areas are they most different?

How exactly to copy these files to your local machine is something I'll leave to you - feel free to ask me questions in class if you can't remember from last week how to do this. I believe in you.

You did it!



(PC: https://www.molecularecologist.com/2015/12/post-holiday-gift-ideas-a-draft-genome/)