Date: 7/23/2024

Author: Anderw Kyle Brand Ardis

Purpose: The purpose of this tutorial is to give a broad overview of the steps involved to generate a fungal annotation using the many tools involved.

Outline

For this specific project, there are fungal samples that have been sent off for sequencing. We desired to:

1. Assemble the Genomes
2. Identify Taxonomy
3. Annotate the Genomes

While there are multiple ways to go about these three steps we have chosen to use the tools for their respective steps

1. MetaWRAP (including megahit, concoct, maxbin2, metabat, quast and checkm)
2. Kaiju
3. Funannotate (including Eggnog, and Interproscan)

Assemble the Genomes

On our BCM cluster I have a metawrap pipeline already established. This pipeline can be copied from a number of locations. If it's needed again in the future, I recommend using the setup found in `/mount/britton/Erika/2024-07-22-Funannotate_life-gift/workflow`. It will be on the leap GitHub soon.

To run these, the first and hardest step is setting up the `nextflow-readfile.csv` where the columns are the sample_ID's, the forward, and reverse reads. After that, the location of the readfile, the work files, and the output files must be updated. Then the job is ready to be submitted.

The above updates sound extremely simple, and they are if you know what you're doing, but it is using nextflow. I would recommend consulting someone before submitting these jobs to ensure they are setup properly.

Once the assmebly is compete the codes found in `/mount/britton/Erika/2024-07-22-Funannotate_life-gift/code` must be run. First the tidy_transfer codes, then clean directory. These are simple codes that will clean things up and move things around so the future files are easier to work with. Consult the leap biobakery-post-processing codes for more details on how or why these codes must be run.

Identify Taxonomy

After the tidy_transfer codes have been run, usually GTDB is used to identify taxonomy, but because GTDB is focused on bacteria, not fungus, kaiju is used. Convienently I've created a code to run kaiju for you, you'll just need to update the input and output locations of the code `kaiju-run.sh` (currently found in code/kaiju). Make sure the output directory location exists. This code will take a few hours to run.

While that is running I recommend seting up `making-checkm-mapping-file.py`, `making-kaiju-mapping-file.py`, and `making-quast-mapping-file.py`. and running the chekcm and quast ones. These will be used later to create a nice table showing completeness and contamination with the kaiju one once that is finished.

Once `kaiju-run.sh` is completed, run `making-kaiju-mapping-file.py`, and if not previously done, run the other two "making-X-mapping-file.py" codes. These are very quick.

an important note. I'm bad at coding and have never been able to drop some of the endings off of the the 'key' column in the output files of making-kaiju and making-quast. so they will not merge together because there will be random ".strict" or ".orig" appended in the 'key' column. Those must be removed. I usually do this in excel by making a new column beside it, calling it 'key' and running `=TEXTBEFORE(A2,".",-1)`. This will return everything before the last period in cell A2. It's a quick easy fix since I can't seem to code better and make these splits in the key without breaking something. If you skip this step, the files will not merge.

Finally you should download all of them and using something like a jupyter notebook, you should merge them into one table. I would recommend using my already created jupyter notebook named `merging_kaiju_to_quast_and_checkm.ipynb`.

If you do not have a jupyter notebook you can use google colab. To You can upload the .ipynb file there, and you'll need to add these two lines at the beginning to upload the files:

```
from google.colab import files
uploaded = files.upload()
```

and at the end you can save the final file with

```
files.download('metagenome_assembly_and_classification.tsv')
```

At this point the saved output file should give you a decent idea of what each bin actually is.

Finally. We will need to get all of the fastas in one place. You could leave them where they are and run the annotation codes by finding the bins one by one, but that is a bit more difficult. I've made a 'for' loop here to make symbolic links to all of the fastas for you to process from here.

In this instance we will make the funannotate directory to keep all of the funannotate work in. and within that we will make fastas.

```
mkdir funannotate
cd funannotate
mkdir fastas
cd fastas
```

go into the fastas folder and run the for loop.

```
for dir in /mount/britton/Erika/2024-07-22-Funannotate_life-gift/workflow-
output/BIN_REASSEMBLY/*; do
  dirname=$(basename "$dir")
  for file in "$dir"/reassembled_bins/*; do
    filename=$(basename "$file")
    new_filename=$(echo "$filename" | awk -F. '{print $1"."$2".fa"}')
    ln -s "$file" "${PWD}/${dirname}_${new_filename}"
  done
done
```

now these symbolic linked fastas will operate normally for the next steps.

Annotate the Genomes

Funannotate is a simple idea, that uses a lot of dependancies. I had extreme difficulty downloading it using conda or mamba. I resorted to singularity.

Singularity, like the more commonly known "Docker", is a platform that allows developers to create, deploy, and run applications in isolated environments called containers, which bundle the application and its dependencies together. This ensures that the application runs consistently across different computing environments, making it easier to develop, test, and deploy software. These containers get a little complicated as well, because they do not always communicate nicely with the rest of the computing system. Luckily I have taken care of most of that for you.

First we need to set it up so it can be run..... before we do that we are going to refer to the very breif tutorial at the bottom on "Screens". Once you understand screens you will need to understand "PATH". Please refer to that very brief tutorial if you're unaware.

Now that you have a screen, grab a node with little memory and a lot of cpus with this line `srun --tasks 1 --cpus-per-task 12 --mem=4G --time 160:00:00 --pty bash` in this you are asking for 12 cpus, with a total of 4G, for 160 hours. So your screen and CPU should be set up for a while. Please remember other people use the cluster, so log out with `exit` as soon as you are done with a node. Once you have a node, update your path. and finally type `module load singularity`

to re-iterate. **make a screen -> grab a node -> update PATH -> load singularity**

We are finally ready to run funannotate.

at any point, please refer to the funannotate tuorial.

first we can run a simple clean with `funannotate-singularity clean -i LG278_bin.1.fa -o practice/LG278_cleaned_bin.1.fa` where LG278_bin.1.fa is the fasta file that we will from now on call <sample_ID>

so the above command is also `funannotate-singularity clean -i LG278_bin.1.fa -o practice/<sample_ID>_cleaned.fa`

Now sorting: `funannotate-singularity sort -i <sample_ID>_cleaned.fa -o <sample_ID>_sorted.fa --minlen 1` For masking the species needs to be known for the first time. `funannotate-singularity mask -i <sample_ID>_sorted.fa -o <sample_ID>_masked.fa -s albicans`

for any of these, you can get a brief description of the commands and options by typing `funannotate-singularity mask --help`, but replace "mask" with whatever command you're most insterested in.

Now for the first command that is going to take a while:

`funannotate-singularity predict -i <sample_ID>_masked.fa -o <sample_ID> --species "candida albicans" --cpus 12` The above command will create a directory by the sample ID. within this directory the most important file for our tutorial will be "<sample_ID>/predict_results/candida_albicans.proteins.fa" This protien fasta file will be used as the input for both eggnog and interproscan, our next two steps.

Running eggnog the much easier of the two:

```
mkdir -p eggnog_out/<sample_ID>/
emapper.py -i  <sample_ID>/predict_results/candida_albicans.proteins.fa -o eggnog_out/<sample_ID>/ --cpu 12
```

Running Interproscan

This part, like eggnog, should also only require updating the <sample_ID> and the species names. But this much more complicated command is actually using its own singularity. Make sure spellings are correct. and paths are correct. There is a lot going on in this one, so take your time. Once you get the first one to run, the rest will be a lot easier.

```
mkdir -p iprscan_output/<sample_ID>
chmod -R ug=wrx,o= *
singularity exec \
    -B /mount/britton/kyle/DBs/interproscan_singularity2/interproscan-
5.68-100.0/data:/opt/interproscan/data \
    -B /mount/britton/Erika/2024-05-28-MAGs-lifegift_fungi/workflow-
output2/BIN_REASSEMBLY/symlink_fastas/copies/practice/iprscan_output:/outp
ut \
    -B $PWD/temp:/temp \
    -B $PWD/<sample_ID>/predict_results:/input \

/mount/britton/kyle/DBs/interproscan_singularity2/interproscan_latest.sif
\
    /opt/interproscan/interproscan.sh \
    --input $PWD/<sample_ID>/predict_results/candida_albicans.proteins.fa
\
    --disable-precalc \
    --output-dir /mount/britton/Erika/2024-05-28-MAGs-
lifegift_fungi/workflow-
output2/BIN_REASSEMBLY/symlink_fastas/copies/practice/iprscan_output/<samp
le_ID> \
    --tempdir /mount/britton/Erika/2024-05-28-MAGs-
lifegift_fungi/workflow-
output2/BIN_REASSEMBLY/symlink_fastas/copies/practice/iprscan_output/temp
\
    --cpu 12
```

The result of this run will give you an ".xml" that you'll need for your final step.

Final step

In this step we combine all of the previous results into a final annotation file.

```
mkdir -p annotation_results/LG80_bin.2
funannotate-singularity annotate --cpus 12 -i <sample_ID>/ -o
annotation_results/LG80_bin.2/ --eggnog
eggnog_out/<sample_ID>/.emapper.annotations --iprscan
iprscan_output/<sample_ID>/candida_albicans.proteins.fa.xml
```

The annotation results will contain the final annotation files of interest.

I hope this tutorial is helpful and clear. Please let me know if you have any questions. Best, AKBA

---

screen Tutorial

Screen is a terminal multiplexer that allows you to create, manage, and switch between multiple terminal sessions from a single window. It lets you run and monitor long-running processes on remote servers without being tied to a single terminal session, as sessions can be detached and reattached later.

first just type "screen " and hit enter to see what happens.

now press, all at the same time, in this order "CTRL (or cmd) + A + D" or "cmd A D". Congrats, you just created, then exited a screen.

Lets create another one, this time we are going to name it. Enter `screen -S learning`. now again "cmd A D".
Now you have started another screen. At this point. Both still exist. To get back into any previously created screen first type `screen -ls`. this will give you a list of previously created screens. The first one should be named 'learning' and the second one will be a random combo of numbers because we didn't name it.

Lets jump back into the 'learning' screen with -R for resume. `screen -R learning`. Now you're back in it. let's enter a quick command. type `this_test="why use screens?". now we just saved "why use screens?" as the variable "this_test". Type` echo $this_test`. Great. As long as you're in this screen that should work.

- Now back out of the screen "cmd A D" Try the same thing `echo $this_test`. Here we can see that the screen is it's own terminal location.

Okay so we have two screens still, type `screen -ls`. To delete them enter the screen (`screen -R learning`) and type `exit`. Poof. it's gone forever. Try to delete screen named with random numbers as well.

Why screens are important

When working on the cluster, specifically running some commandline jobs you need it to keep running for hours and hours.... But you can't just sit there and leave your laptop open for hours and hours. When you run screen you can exit at any point, but the screen will continue you run. When running some of the funannotate commands you will definitely want to use screen to leave it running in the background, and just check up on it later on. There is no limit to the number of screens you can run, and there are no limitations to what you can name them. My current screens when I type `screen -ls` are named "eggs", "ipr", "fun" and "stupid". I usually try to name them something that's easy to remember what is happening on that screen. for example eggs is where I ran most of my eggnog jobs, ipr the Interproscan jobs, fun was the funannotate jobs. and stupid was where I was being stupid trying to do too many things at once.

One final tip. You cannot scroll up and down the same way on a screen. well you can, but first you must type `cmd A [`. When you're done scrolling just close the bracket with `]` and it will jump back to be ready to continue typing.

PATH tutorial

The PATH is an environment variable that tells your computer where to look for executable programs. It lists directories that the system searches to find commands and run applications.

type `echo $PATH`. your PATH is a very important variable, so we always want to be careful when updating it. Notice how each path end is ending in a colon. To update path all you need to do it rewrite the variable. like this:

`PATH="/mount/britton/kyle/DBs/singularity:$PATH` if you enter that exactly, all of the things in "/mount/britton/kyle/DBs/singularity" are now executable from anywhere, which you will need.

IF YOU MESS THIS UP YOU CAN SERIOUSLY MESS UP A LOT OF THINGS. jk. your PATH resets every time you log in. There are ways to update it every time you log in, but we dont't need that for now. Within each of your screens you'll need to, after you grab a node, update your PATH with these lines.

```
PATH="/mount/britton/kyle/DBs/singularity:/mount/britton/Erika/databases/gene_mark/gmes_linux_64_4:/mount/britton/kyle/DBs/github_repositories/eggnog/eggnog-mapper-2.1.12:/mount/britton/kyle/DBs/github_repositories/eggnog/eggnog-mapper-2.1.12/eggnogmapper:/mount/britton/kyle/DBs/github_repositories/eggnog/eggnog-mapper-2.1.12/eggnogmapper/bin:$PATH"
```

just copy and paste that every time. be careful that you do not have any spaces or enters between, it should be one long line with **no spaces**. (I would recommend copying from the README.txt, the .pdf tends to copy over spaces)