Microbiome dataset processing

This document shows an example workflow of how to process a dataset. This workflow is heavily geared towards datasets pulled from Qiita, as their files have already been preprocessed extensively. Use this document in conjunction with the other resources in Lab Docs to provide a more in-depth look at the steps necessary to process a dataset for storage on Blackbird or another lab computer.

Color coding:

optional steps are highlighted yellow

unnecessary steps are highlighted red

if a step is not highlighted it is considered important

Step 1: Creating directories and downloading files

In this step you will create necessary directories and download and move files into those directories. You don't necessarily need to follow the following commands exactly, if you have familiarity with linux/unix command line interface (CLI) you may perform these steps in whichever order you prefer. The important thing is to try to have some consistency and standards across our datasets on our computers in terms of file structure and organization. This will make it easier for everyone in the lab to access and utilize these datasets.

Make the dataset directory (let's call it "hmp2" in this example).

```
mkdir hmp2
```

Download the metagenomics .biom files.

Move all downloaded .biom files to hmp2 directory.

```
mv ~/Downloads/*.biom hmp2
```

Navigate to the hmp2 directory.

```
cd hmp2
```

Create metagenomics ('mgx') directory within hmp2 directory

```
mkdir mgx
```

Move all .biom files from hmp2 directory into mgx directory

```
mv *.biom mgx
```

Create 16s directory within hmp2 directory

```
mkdir 16s
```

download the 16s .biom files and fasta (.fa) files

mv ~/Downloads/*.biom 16s

move downloaded 16s .biom files directly to 16s directory within hmp2 directory

mv ~/Downloads/*.fa 16s

move downloaded 16s fasta files to 16s directory

cd 16s

nano readme.txt

navigate to 16s directory and create readme file with details like date, where on qiita they were pulled from, how they were processed, etc.

download the metadata.txt file

cd ..

mv ~/Downloads/<metadatafilename>.txt .

navigate to hmp2 directory and move the metadata file into that directory

Step 2: Generating your metadata file

The full metadata file which was downloaded previously may contain significant amounts of information which is not relevant to your research interest. It is possible to derive a smaller metadata file which contains only the relevant information. The original metadata file should be left in place in the main directory for that particular dataset, from there you can use the following commands to pull whichever columns you like. The important command in this step is **cut**. You can find more information about this command here, or by typing **man cut** into the terminal. **man** followed by any linux command will show you the 'manual page' for that command.

less <metadatafilename>.txt

check out what the metadata file looks like. Oftentimes it doesn't make sense to use the terminal to inspect large text files. You can use excel or some other program to view a large table more easily.

open the metadata file in excel

copy the header row

past the row into a new sheet with transpose turned on (paste special)

with this you can find the index numbers of whatever metadata columns you wish to keep

in our case, we just kept 'diagnosis' which was index 42

cut -f 1,42 <metadatafilename>.txt > metadata.tsv

takes the 'diagnosis' column (index 42) from the metadata.txt file and outputs that column to metadata.tsv. This command will change depending on what columns from your metadata file you deem relevant. This is just an example of how to take one column from the original metadata file and make a new .tsv file with that column.

less metadata.tsv

cat metadata.tsv

inspect metadata.tsv to see if it looks good

Step 3: Merge BIOM files

For this step and several subsequent steps you will need to have woltka installed. It is recommended you use mamba or conda to make an environment with the necessary packages. This step may not be necessary if your dataset contains a single BIOM file.

cd mgx

navigate to the metagenomics directory

woltka merge -h

with woltka installed, this will show you what it can do, -h is generally the help command

woltka merge -i <file1_none.biom> -i <file2_none.biom> -i <file3_none.biom> … -o merged.biom

merge the separate .biom files into 1 file. need -i before each input file, set output file with -o. The exact usage of this command will differ depending on your dataset.

ls -l

check that the merge worked and eyeball the size of the merged file

biom

opens up the biom program page so you can see what options there are

biom summarize-table -h

opens the help page for the summarize-table function of biom

biom summarize-table -i merged.biom

perform summarize-table function on merged biom file. This step is simply to check and see what the merged BIOM file looks like.

biom summarize-table -i merged.biom –observations | less

pipes the output from summarize-table function with –observations option into less, which displays info. Similarly to the previous command, this is for basic checks to see if everything looks sensible and to get some basic info on your merged file.

Step 4: Identify matching IDs between metadata and data

Here you are interested in making the sample IDs from the metadata and the sample IDs from the merged BIOM file (your actual data file) match. In this example, there were some samples in our metadata which were not present in the data file. This may be the case with your dataset, or it may not be, but the following commands should give you some idea of how to handle the situation, whichever it may be. Again, this document is more of a loose guide than a definitive order of operations, and you may encounter situations which are different than those displayed here. **Note:** We don't actually do anything with the joined_ids.txt file from this step, so you don't necessarily need to do this step. It is pretty straightforward to achieve this in a jupyter notebook, although if you take care of it here, your notebook will look cleaner.

biom subset-table -h

brings up subset-table help page

```
cut -f1 ../metadata.tsv | tail -n+2 > ids.txt
```

takes the ids from the metadata file and puts them in the ids.txt file

```
biom subset-table -i merged.biom -a sample -s ids.txt -o merge_sub.biom
```

this performs the subset table to try to get only the ids present in the metadata to be in the merge_sub.biom file. This didn't work as intended, because there were sample IDs in ids.txt which were not present in merged.biom. You can't make a subset based on ids.txt if all of those IDs are not present.

```
biom table-ids -h
```

displays help page for table-ids function of biom. Here the motivation was that we could get the sample IDs of the merged.biom file using the table-ids command.

```
biom table-ids -i merged.biom > table_ids.txt
```

gets the ids from merged.biom and puts them in table_ids.txt

```
less table_ids.txt
```

sanity check, views the content of table_ids.txt

```
wc -l table_ids.txt
```

returns number of lines (ids) in table_ids

```
cat table_ids.txt | wc -l
```

more robust(?) version of previous command

```
wc -l ids.txt
```

returns number of lines (ids) in ids.txt. Compare the number of lines in ids.txt to the number of lines in table_ids.txt. At this point we were interested in finding the intersection of ids.txt and table_ids.txt

```
join <(sort ids.txt) <(sort table_ids.txt) > joined_ids.txt
```

the commands in the parentheses are executed first, then the output from those is passed as input to the join command, and the output from that is saved as joined_ids.txt, which is the intersection between both sets. So joined_ids.txt has all the sample IDs which are present both in our metadata and data (BIOM) files.

```
wc -l joined_ids.txt
```

check the number of ids in the intersection. Can compare the output of this command to the output from the previous two **wc -l** commands to see how the **join** command worked.

Step 5: Filter BIOM file by read count

For the processing we have performed thus far, we chose to filter out any features (species) which had less than 100,000 total read count. All datasets don't necessarily have to be filtered in this way, but it is an easy way to clean up the data for downstream analysis.

Check the summarize-table again.

```
biom summarize_table -i merged.biom | less
```

We will use Woltka to perform filtering. Type `woltka` to display the command-line interface (CLI) of the program. Type `woltka filter -h` to display the help information of the filtering function.

We were interested in filtering any features who had total abundance counts less than 100,000.

```
woltka filter -i merged.biom -c 100000 -o merged_filt.biom
```

this didn't do what we want, because this filters samples, not features

rm merged_filt.biom

removed the unnecessary file

biom summarize-table -i merged.biom > test.txt

wanted the summarize-table in a .txt file. This gives us test.txt which contains each feature and their total counts.

less test.txt

looked inside the test.txt file to identify the feature below which all other features had counts <100,000. This is a manual process. You scroll through test.txt until you find the first feature with greater than 100,000 total read count. Copy this feature name to use in the next command.

grep -A 9999 “11484.HSM7J4KA” test.txt > out.txt

this is how we filtered everything which had counts below <100,000. The id “11484.HSM7J4KA” was found manually in the previous steps. grep is a command which searches a file for a particular pattern of characters, and returns all the lines containing that character. The we’ve used it here is by taking advantage of the **-A** option. What this command does is that it searches test.txt for the pattern “11484.HSM7J4KA” or whatever your relevant feature name is. Once it finds this pattern, it will return the line containing that pattern plus the next 9999 lines in the file. Because there are less than 9999 lines (IDs) in test.txt, it will just return the pattern it identified and everything below it. This works because test.txt has the IDs sorted in ascending order, so the IDs with the lowest count will be at the top of the file, and IDs with higher counts will be lower in the file.

less out.txt

confirmed that the filtering works

tail out.txt

further confirmation that filtering worked

wc -l out.txt

returns number of ids in filtered set

cut -f1 -d ':' out.txt > new_out.txt

removed everything before the colon (:) in the out.txt file and put the result into the new_out.txt file. This command removes the count values from the file, so all that is left in new_out.txt is the ID.

less new_out.txt

wc -l new_out.txt

inspection of new_out.txt

biom subset-table -i merged.biom -a sample -s new_out.txt -o final_filtered.biom

using new_out.txt gives us a final_filtered.biom file which contains only features which have greater than 100,000 total abundance counts.

less merge_sub.biom

inspect the filtered biom file

biom convert --to-tsv -i <inputfile>.biom -o <outputfile>.tsv

converts biom to tsv file

tail -n+2 <inputfile.tsv> > <outputfile.tsv>

removes first row from file, useful after doing convert to remove “converted from biom file” line

Note: an easier way is: `sed -i 1d table.tsv`

Step X: Drop invalid sample IDs

Some samples should not be included in an analysis. For example, sample IDs that include the word “blank” usually represent blank controls. These sample only carry background noise, and can significantly impair the analysis result.

Protocol 1: Filter the BIOM table

First, get a list of sample IDs in the BIOM table:

```
biom table-ids -i table.biom > ids.txt
```

Drop invalid sample IDs. For example, to exclude sample IDs that contain the word “blank”, one can (`-v`: negative selection, `-i`: case-insensitive):

```
cat ids.txt | grep -v -i blank > ids_filt.txt
```

Filter the BIOM table to include only valid sample IDs:

```
biom subset-table -i table.biom -a sample -s ids_filt.txt -o
table_filt.biom
```

Protocol 2a: Only filter the metadata table (`AWK` version)

The following command uses the AWK program to achieve pattern matching in a specific column (`$1`: first column, `~`: if it contains a word, `!`: negative matching, `tolower`: convert to lower case):

```
awk 'tolower($1) !~ "blank"' metadata.tsv > metadata_filt.tsv
```

This command works like a magic. The downside is that AWK is more complicated than grep and is harder to learn.

Protocol 2b: Filter the metadata table (`grep` version)

In a metadata table, sample IDs are included in the first column, from the second row. Therefore, one should first copy and paste the first row:

```
head -n1 metadata.tsv > metadata_filt.tsv
```

Then get a list of sample IDs:

```
tail -n+2 metadata.tsv | cut -f1 > ids.txt
```

Drop invalid sample IDs:

```
grep -v -i blank ids.txt > ids_filt.txt
```

Here is the critical part: Filter the metadata table body such that it only contains valid sample IDs. Note that for `join` to work, both sides have to be sorted, therefore the outcome does not preserve the original order. (`-j1`: join based on the first column, `-t$'\t'`: columns are separated by tab, `sort -k1,1`: sort by the first column).

```
join -j1 -t$'\t' <(tail -n+2 metadata.tsv | sort -k1,1) <(sort
ids_filt.txt) >> metadata_filt.tsv
```

After executing protocol 2, one can filter the BIOM table based on the filtered metadata table. Refer to “Step 4: Identify matching IDs between metadata and data” for the method.

Step Y: Strip Qiita study IDs

In Qiita, sample IDs are named like “10317.000001062”, in which “10317” is the study ID and “000001062” is the unique sample ID in this study. However, such format may complicate subsequent analyses. For examples: Excel may incorrectly parse this string as an integer; R may add an “X” before each sample ID that starts with a digit.

Therefore, an elegant method is to replace “10317.000…” with “S” (meaning “sample”) or something else that one prefers. As such, “**10317.000001062**” becomes “**S1062**”. To achieve this, one can do the following.

For the metadata table, one can use the following `sed` command (`s///`: substitute, `^`: beginning of a line, `*`: zero to multiple number of cases).

```
sed 's/^10317\.0*/S/' metadata.tsv > metadata_edit.tsv
```

For the feature table, it is much easier to do it when it is in TSV format (`\t`: tab, `g`: substitute all matches):

```
head -n1 table.tsv | sed 's/\t10317\.0*/\tS/g' > table_edit.tsv

tail -n+2 table.tsv >> table_edit.tsv
```

If the feature table is in BIOM format, there is probably not a Linux way to achieve this. One will need some Python coding to do it. Below is a Python script **`strip_qiita.py`**:

```
from sys import argv
from biom import load_table
from biom.util import biom_open

# helper function that converts a sample ID
def _convert(id_):
   return 'S' + id_.split('.', 1)[1].lstrip('0')

# read a BIOM table
table = load_table(argv[1])

# get sample IDs
ids = table.ids()

# generate a mapping of old to new IDs
id_map = {x: _convert(x) for x in ids}

# convert old IDs to new IDs in the table
table.update_ids(id_map=id_map)

# write updated table
with biom_open(argv[2], 'w') as f:
   table.to_hdf5(f, table.generated_by)
```

Use the Python script like this:

```
python strip_qiita.py table.biom table_edit.biom
```

Step Z: Convert cell values to integers

In a TSV file exported from BIOM, cell values are always floating-point numbers. However, a typical feature table in microbiome research only have integers, i.e., the numbers of sequences assigned to individual features. To save disk space, and to simplify data analysis, one can remove the suffix `.0` from each cell value with:

```
head -n1 table.tsv > table_int.tsv

tail -n+2 table.tsv | sed 's/\.0\t/\t/g' | sed 's/\.0$//' >> table_int.tsv
```

Check whether there are remaining decimal points, which indicate non-integer cell values (and in which case you should not cast all numbers into integers):

```
tail -n+2 table_int.tsv | grep '\.'
```

The end.