UNL - QIIME Tutorial

Authors: JJM.Riethoven[1)], R.Sinha[2)]
1) BCRF, Center for Biotechnology, University of Nebraska-Lincoln 2) Food Science and Technology, University of Nebraska-Lincoln

Acknowledgements: data sets used in this tutorial are courtesy of Drs. Daniel Schachtman (sorghum root; startup funds from ORED/ARD and Dept. of Horticulture and Agronomy) and Devin Rose (human gut; Wheat Innovation Fund/Nebraska Foundation).

Last update: 19 October 2016.

1 Introduction

The tutorial today will give a high level overview of some of the steps that are involved in the analysis of metagenomics data. We will be using the QIIME software, mostly, for the analysis steps involved in this tutorial. Due to time-constraints we also have prepared input files that can be analyzed in a reasonable amount of time (i.e. these are not complete data sets).

During the tutorial we will be using Mozilla's etherpad – some of the longer commands will be posted there so you can cut&paste these to avoid typing errors. You can also use the Etherpad to ask questions or help with steps in the tutorial. You can access the Etherpad by pointing your browser to:

https://public.etherpad-mozilla.org/p/UNLQiime_Oct2016

(or shorter: http://go.unl.edu/qiime)

1.1 Requirements checkup

In short, you need:

1. An ssh client (e.g. terminal on Mac, PuTTY on Windows)
2. A file transfer program (preferably Globus as you have used it this morning)
3. An account with the Holland Computing Center, and DUO authentication set up

If any of these are currently not working, please let us know immediately.

1.2 Logging in to the Holland Computing Center

You will either connect via PuTTY (Windows) or an ssh command in a terminal (Mac, Linux).

1. PuTTY. Important settings are hostname "crane.unl.edu" and connection type 'SSH'.
2. SSH command (on Mac): `ssh yourusername@crane.unl.edu`

Let us know if you have trouble logging in.

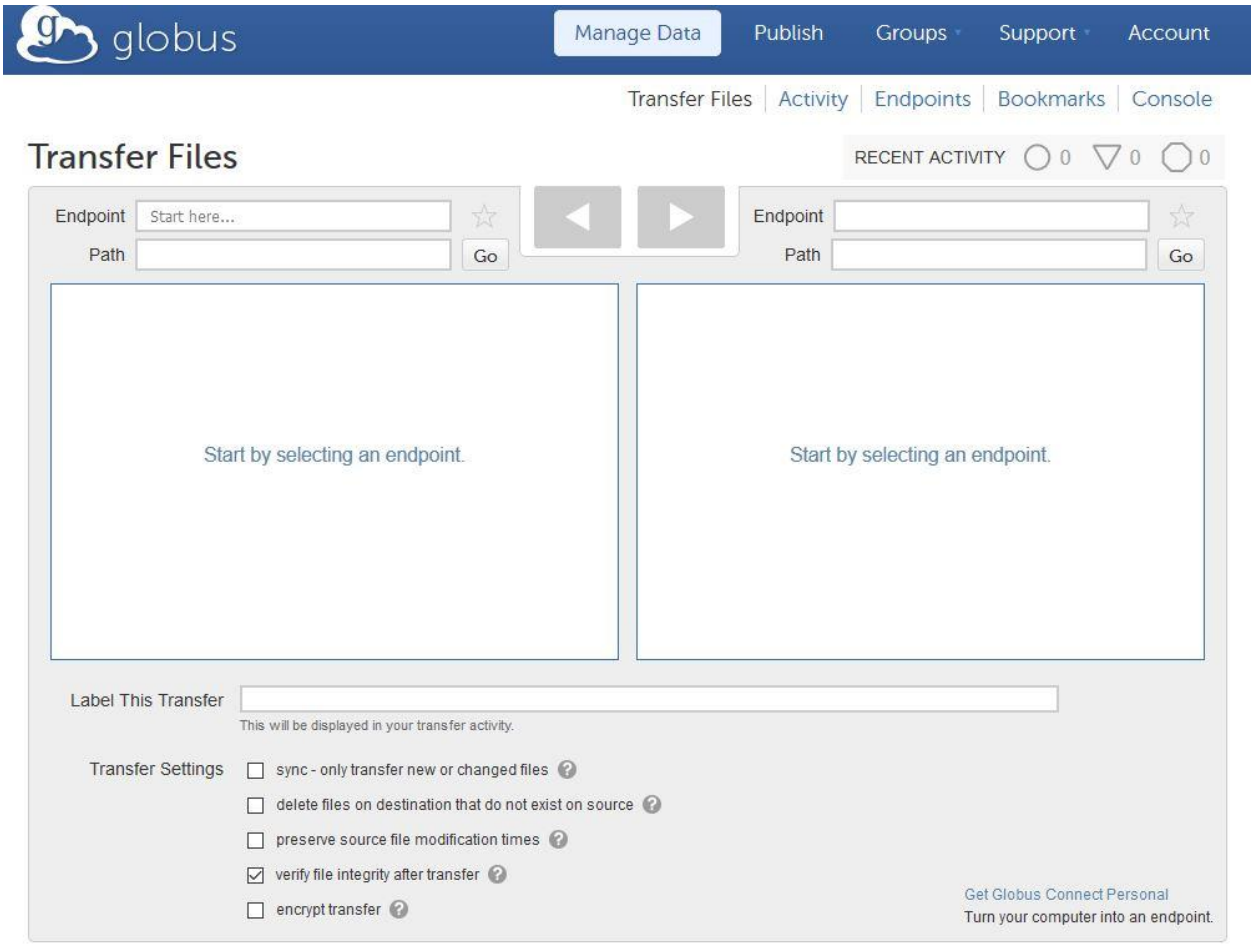
1.3 File Transfer

During the tutorial we will transfer some files from the compute cluster (crane.unl.edu) to our desktop. There are really several ways to do this, but we prefer you to use Globus. Before you continue, make sure your Globus Personal Endpoint is set up on your laptop and running (https://www.globus.org/globus-connect-personal)

First, let's make a simple file on crane.unl.edu to test this. Log on to crane.unl.edu (PuTTY or ssh), then type:

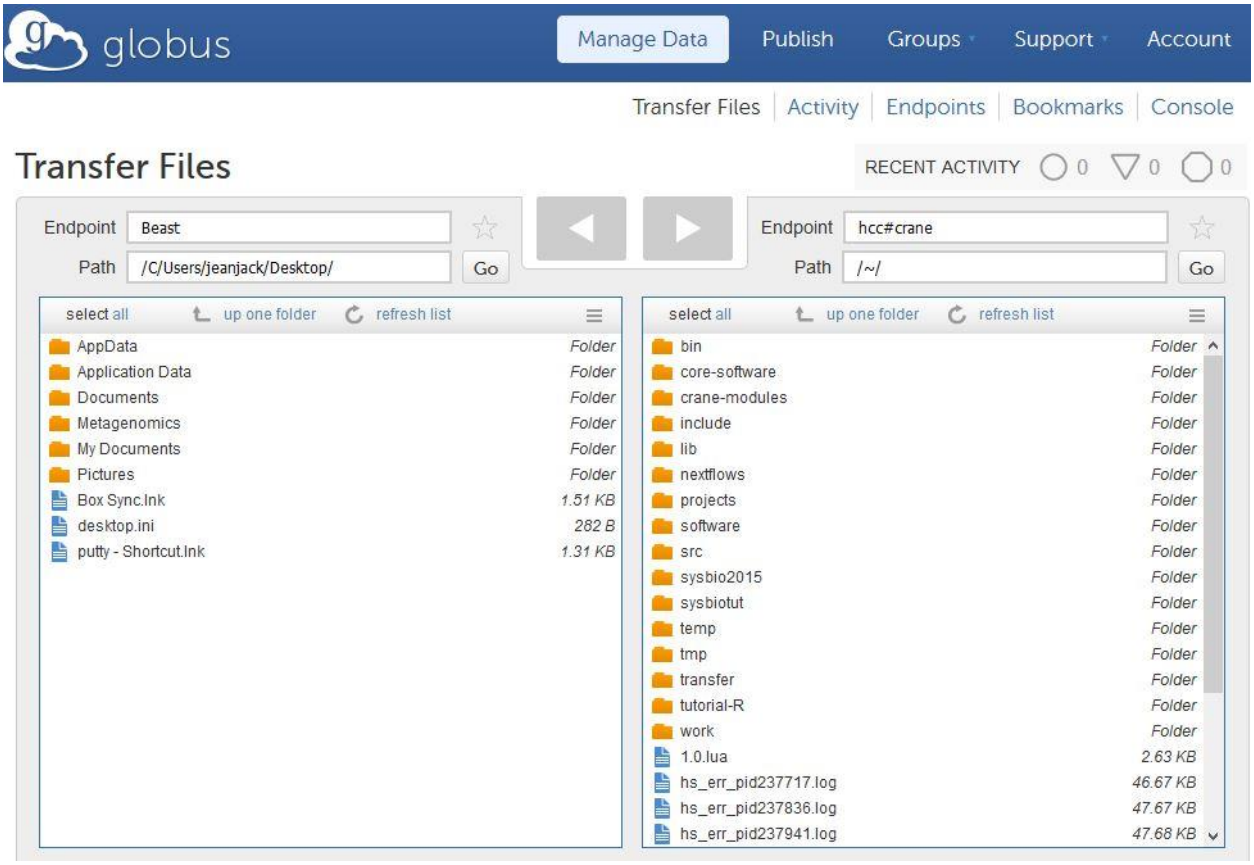
```
echo "this is a simple file" > myfirst.txt
```

In your browser, navigate to https://www.globus.org/app/transfer (you might have to log in). The "Manage Data" tab should appear as shown below. As explained this morning, this is one of the main views that you will use to transfer data within HCC, between your desktop or laptop to HCC, and vice versa.



We are going to use it now to copy the file we just created from crane.unl.edu to your laptop. In the right pane, choose as endpoint hcc#crane, and once chosen it should automatically move to your home directory (designated as /~/). Depending on whether not you have been active previously on crane, you might see a lot of files and directories. The file 'myfirst.txt' should appear as well.

In the left pane, let's connect to your personal Globus endpoint. Use the name that you have set up in the morning. It will also display the content of the folder that it deems your 'home' on your laptop (on Windows this tends to be your 'Documents' folder). For ease of use during this workshop, please navigate to your desktop (Use 'up one folder' and then drill down). On a Windows machine your desktop is (normally) located at C:/Users/<your_username>/Desktop, on a Mac it is located on /Users/<your_username>/Desktop.



Now, on the left pane, find and select (left-click) your 'myfirst.txt' file. One of the big arrows on the top will light up blue, and you can click it to start the copying process from crane to your laptop's desktop. Once it is completed (you get an email and you can see this in the 'Recent activity' panel, look on your desktop and your text file has appeared.

In a couple of minutes, we will come back to Globus to copy the data and scripts required for this workshop to your work directory on crane, so keep a tab open on your browser pointing to Globus.

1.4 Setting up tutorial environment

During the tutorial we will be doing all the analyses in 'worker nodes' on the HCC cluster. Normally, you will have to write simple job scripts containing your commands and submit these to the job scheduler on the compute cluster. Specifically, for this workshop, we will use interactive commands but we still need to request the resources we need. To do so, type the following exactly as shown:

```
srun --pty --time=5:00:00 --mem-per-cpu=5G  --reservation=qiime --
ntasks=2 /bin/bash
```

(or copy from Etherpad)

That should move us to a worker node for all further analysis[1].

1.4.1 Working directory

Home directories are read-only for performance reasons on worker nodes, so we need to move to a different location. There is a special location where we have high-performance file access to our data: /work. To access your work directory, you will need to know your primary group on the HCC cluster, and your username. The final location is: /work/yourgroup/yourname. For example, /work/riethoven/jeanjack is the presenter's working directory. Let's move to your own working directory via the change directory command 'cd' (change riethoven and jeanjack into your own group and username):

```
cd     /work/riethoven/jeanjack
```

If you type 'pwd' (print working directory) it should show the same location. If not, please ask for help.

You can also use a shortcut via an environment variable: $WORK which points automagically to your work directory.

```
cd    $WORK
```

Once we are in our working directory, let's make a directory that we will use to do the tutorial and move to it.

---

[1] All steps in this workshop are also available for normal job scheduling on crane.unl.edu: you can submit the 'shortcut' scripts via the sbatch command. Don't do this now as we will run all analyses interactively, but if you want to rerun this workshop at a later time the above 'srun' command won't work and you will have to submit your scripts. For example:

```
sbatch step_1.sh
```

```
mkdir qiimetut

cd qiimetut
```

Now, move back to your browser. Either click on this link http://go.unl.edu/qiimetut, copy/paste, or type it in your browser. It should bring you to Globus again, and the left pane should point to an endpoint called 'QIIME Tutorial'. In the right pane, we want to point to the directory we just made. Use hcc#crane as endpoint, but change the path to /work/yourgroup/yourusername/qiimetut (replace yourgroup and yourusername with your own).

Your qiimetut folder in Globus should be empty, and the left panel should show you various folders. Select all the folders and files in the left panel and press the arrow to start the data copy to your work qiimetut folder. The copy should take a few minutes, maybe a bit longer since we are all copying at the same time.

If you ask for a file listing, you should see something similar like this (actual content might differ slightly):

```
ls -hl

total 472K
-rw-r--r--  1 jeanjack riethoven  129 Oct 13 11:56 CHANGELOG
-rw-r--r--  1 jeanjack riethoven  230 Oct 10 13:18 CONTRIBUTING.md
drwxr-xr-x  4 jeanjack riethoven 4.0K Oct 13 10:54 raw
-rw-r--r--  1 jeanjack riethoven   64 Oct 10 13:18 README.md
drwxr-xr-x  2 jeanjack riethoven 4.0K Oct 13 11:56 scripts
```

If you don't see at least a raw and a scripts directory please let us know.

That concludes setting up for the tutorial. Let's move on to the real work.

2 Raw sequence

In this workshop, we are providing two partial sets of data: a sorghum root microbiome and a human gut microbiome, courtesy of Drs. Daniel Schachtman and Devin Rose, respectively. The data sets have been downscaled to about 10% of their actual size, to make processing it during the workshop a lot faster. The 16S sequencing for these projects were done pair-ended so we will have two files (forward and reverse) for each sample. All initial files are in the FASTQ format, and have been demultiplexed (i.e. they have been divided by sample, and barcodes have been removed).

The raw files are located in the 'raw' directory. We'll have a look at them later, but first you must make a choice which data set you want to work with as we only have time to go through one set this afternoon. Move to the raw folder:

```
cd raw
pwd
```

(pwd should display, at the end of the line, qiimetut/raw. If not, please ask for help).

Then, make a choice. If you want to work with sorghum, type:

```
ln –s sorghum data
cd ..
ln –s sorghum_map.txt map.txt
```

If you want to work with the human gut instead, type:

```
ln –s human data
cd ..
ln –s human_map.txt map.txt
```

The result will be that we have created a soft link named 'data' that will point to the folder of your choice (and a link to a mapping file which we'll discuss later). Type:

```
ls –lh raw
ls –lh raw/data/
```

and you should see both that data points to the folder you picked, and gives you the content of the data folder. All subsequent analyses are done on the 'data' folder.

Let's move back to the main qiimetut folder:

```
cd $WORK/qiimetut
pwd
```

The next step is to prepare some scripts for the workshop; this step is only required for this workshop and is not a part of the normal workflow of QIIME. Type and run:

```
./scripts/prep_scripts.sh
```

There should be no warnings or errors – if there are please let us know.

3 Quality assurance and filtering

An integral part of any next-generation sequencing analyses is to assure that your data is of high-quality. Many factors can influence the quality, and poor quality and errors will adversely affect the final outcome of our work.

Once simple tool to quickly check for problems is FastQC. It will check the sequence and provide us with statistics on the quality. FastQC has both a graphical user interface and a command-line version. We will be using the command-line version.

First load the environment that is required for FastQC:

```
module load fastqc
mkdir –p qa
```

And then run it on our data files:

```
fastqc  raw/data/*  -o qa/
```

(please note that file names and commands are case-sensitive)

After running these commands, you should have several zip and html files in the 'qa' directory (ls –lh qa). To look at the content of the .html files, use Globus to transfer them to your desktop and then double-click the html files from within Windows Explorer or Finder.

16S sequencing is quite different in the resulting files than for example the mainstream DNA or RNA-seq, which tends to be only one or a few species. We have to keep this in mind when we look at the results of the quality control. We also need to reminder that we are only looking at 10% of the number of sequences here. Pick any report to view, and let's look at the 'Per base sequence quality' graph. You will see that the base-by-base quality scores get progressively gets worse towards the end of the sequence. Good (Phred) scores for genome assembly or SNP detection are 30 and above. We will do 'sort of' an assembly when we start joining and clustering the data later in the pipeline. As you can see, there are quite some sequences that trail below this cutoff. That is a problem, and we'll tackle that in the next section.

There are other graphs and sections that do not look good (e.g. the per base sequence content), but normally we look at this when we have millions of reads, not the roughly 10,000 that we have now. The other section is the overrepresented sequences, but again, keep in mind that we are actually sequencing a very small part of the 16S ribosomal gene, which tends to have very conserved stretches between species and families.

For some of the downstream analysis we also need to know what the quality score encoding in our fastq files is. Most newer sequencing files are encoded in Phred33, and a lot of the computational tools will, by default, assume that Phred33 encoding is used. There are some exceptions to the case where tools check the encoding first, but without knowing for each tool if they are doing this we want to know, upfront, what the encoding is so we can direct tools to use this.

```
module load ea-utils
```

```
find raw/data/ -name "*.fastq" -exec fastq-stats \{\} \; | grep -i
phred
```

Now we know that all fastq files are encoded Phred33, instead of older encodings such as Phred64.

3.1 Filtering

We are going to improve the data we have by filtering out the 'junk' on the reads. For some reads we only needs to remove a few bases at the 5' or 3' end, others will be so bad throughout that we are going to throw them out completely. There are many tools available for read trimming – but we cannot just pick any. One important aspect for later in our pipeline is that we keep the forward and reverse reads in sync, as the tools expect they are so and do not check this. There are additional tools that can 'fix' these discrepancies, but we have selected for this tutorial a software package that does all this for us: trim_galore (http://www.bioinformatics.babraham.ac.uk/projects/trim_galore/)

The template for the command line is:

```
module load python/2.7
module load cutadapt
module load fastqc
module load trim_galore
```

Trim_galore uses a software tool called 'cutadapt' to check for, and remove, any residual sequencing adapters. Cutadapt specifically requires python version 2.7. Trim_galore also can use FastQC to do an automatic round of quality checking – we are going to make use of that.

```
trim_galore -q 20 --length 40 --paired  --retain_unpaired   --
fastqc_args "--outdir OUTPUT_DIRECTORY_FASTQC" -o
OUTPUT_DIRECTORY_FILTERED FORWARD.fastq REVERSE.fastq
```

It is a complicated command line, but four places are of interest: OUTPUT_DIRECTORY_FASTQC will be the name of the directory where FastQC will put its quality control files that were run on the post-filtering fastq files; OUTPUT_DIRECTORY_FILTERED is where the output files of the filtering and trimming process will be, and FORWARD.fastq and REVERSE.fastq are the names of the pair-end fastq files for a particular sample.
Trim_galore and the cutadapt tool will also use the Phred33 encoding by default, so we don't have to mention this here. If our files were encoded with the older Illumina Phred64, we would have added –phred64 to the command line.

An example command-line with all information filled in is:

```
mkdir -p filtered/84_S69_L001

trim_galore -q 20 --length 40 --paired  --retain_unpaired
--fastqc_args "--outdir qa/" -o filtered/84_S69_L001
raw/data/84_S69_L001_R1_001.fastq  raw/data/84_S69_L001_R2_001.fastq
```

We have a convenience script that does this for all our input fastq files:

```
./scripts/filtering.sh
```

If we compare the raw sequence file sizes with the new file sizes:

```
ls -lh raw/data/*.fastq
ls -lh filtered/
ls -lh filtered/125_S99_L001
```

(Replace the last part of the third 'ls' command with a folder name you see – the example here is for human gut data)

We'll see that trim_galore did its work and removed bases and reads. It wrote the individual forward and reverse reads that are still paired up into files with a 'val_1' and 'val_2' keyword. Any orphan reads are written with the 'unpaired' keyword in the filename. We can also check the quality reports by looking in the 'qa' directory since trim_galore ran FastQC for us.

Transfer the html files from the 'qa' folder that have the keyword 'val_1' and 'val_2' (forward and reverse reads, respectively) in them back to your laptop and view them. Compare with the previous, raw, files.

Note: although it is great that trim_galore filtered and trimmed the fastq files for us, the names with the additional val_1 and val_2 won't work for us later in the QIIME pipeline as QIIME expects the file name to be the same with the exception of _R1 and _R2 for read 1 and 2, respectively. We need to run a little script to rename the val_1 and val_2 files:

```
cd $WORK/qiimetut
pwd

./scripts/fix_filtered_names.sh
```

After running this, no file with a keyword val_1 or val_2 should be existing in on the of sample directories.

4 QIIME

Now that we have clean and high-quality sequence reads, we can continue with the QIIME computational pipeline. Please note that we have here changed some of the normal QIIME workflow, as QIIME normally joins reads and then does internal quality control. We have put quality control and filtering ahead in the workflow, as we can then step in if there are artefacts that normally wouldn't be detected.

Depending on the size of the insert (i.e. the area you sequence in 16S), we might need to join reads if the insert is small as compared to the reads, or not, if the insert is long and/or the reads are short. For both the data sets we expect a significant overlap: the sorghum data set used the V4 region (about 250bp) with 2x300bp reads, and the human gut data set used the V3V4 region (about 560bp) with 2x4300bp. In this case, we expect, and should require, reads to overlap.

In QIIME there are two ways to join paired-end reads: either with join_paired_end.py or with multiple_join_paired_ends.py. The first command tends to be used if you have just two big FASTQ files with all samples included therein, complete with barcodes included within or in a separate file. This is not a normal way of delivering data from sequencing centers nowadays; they should demultiplex your data (check barcodes, remove them, and bin your sequence data by sample).

We will use the second method, meaning you probably have hundreds of files nicely separated by sample. For this approach, to join reads, we will use multiple_join_reads.py.

In its simplest form, you can just point the command to a directory which holds all your paired-end fastq files and it will join them for you, putting a joined read file and possible two un-joined read files into new folders named by sample.

We will do this, but we want to make sure all new, joined, files, are within a new folder. Let's make this folder first:

```
mkdir –p join
```

We also need to get rid of all files that contain bad sequences (from our previous step). We can either move them from the 'filtered' folder to somewhere else (if you really want to keep them), or we just flat-out delete them. In this tutorial we'lll do the latter:

```
find filtered/ -name "*_unpaired_*" | xargs rm –f
```

Now our filtered folder only has the good stuff: run the commands:

```
module load qiime

multiple_join_paired_ends.py -i filtered/ -o join/ --read1_indicator
_R1 --read2_indicator _R2
```

(-i points to the input directory, and –o to the output directory).

After this, we have sixteen new folders in the 'join' directory, named after our samples.

```
ls –lh  join
```

Check one of those folders by:

```
ls –lh  join/125_S99_L001_R1_001
```

(Replace 125_S99_L001_R1_001 by another folder you see if you are working on sorghum data).

You should see something similar as this:

```
total 7.2M
-rw-r--r-- 1 jeanjack riethoven 5.9M Oct 19 14:07 fastqjoin.join.fastq
-rw-r--r-- 1 jeanjack riethoven 694K Oct 19 14:07 fastqjoin.un1.fastq
-rw-r--r-- 1 jeanjack riethoven 667K Oct 19 14:07 fastqjoin.un2.fastq
```

We are going to use the joined fastq file. As you can see, unfortunately the file is not named after its sample, something which is required for later steps in the analysis, specifically the mapping file where we denote each file name by its sample and condition/treatment parameters. So, we are going to do another little step to fix this, and run a script that will 1) convert fastq files into fasta files (which QIIME needs in later steps), 2) rename the fastqjoin.join.fastq file into its sample name which we will derive from the parent folder. We also remove the unjoined files at the same time. Run:

```
./scripts/fix_joined_names.sh

ls -lh join/
```

You'll see the obscure join file has been renamed to something that can be used to distinguish between samples.

At this point we have clean, joined, reads, each in its own fasta file. Now we need to explain to QIIME which files belong to which genotype/treatment/condition combination. We normally explain this to QIIME in a mapping file, where barcodes/primer sequences are linked to samples, or, in our case, where we link samples (and their meta_data) against file names.

Have a look at the mapping file, map.txt:

```
less map.txt
```

(press 'q' to exit it). The important columns here are #SampleID, the metadata fields which describe our individual samples, and the InputFileNames column which denotes the file name that should be linked to this sample. We'll use the following commands to add sample ID's straight into the fasta files, and to concatenate the fasta files into one big file.

```
mkdir -p combined
add_qiime_labels.py -m map.txt -i join/ -c InputFileNames -o combined/
ls -lh combined
less combined/combined_seqs.fna
count_seqs.py -i combined/combined_seqs.fna
```

4.1 OTU picking

The next step would be to cluster similar sequences into Operational Taxonomic Units (OTUs); with the premise here being that sequences that look alike (i.e. align with few substitutions or indels) likely come from the same species, genus, or higher taxonomic unit. There are three widely used techniques of clustering: de novo, closed-reference, and open-reference OUT picking. With de-novo we only allow

reads to align against other reads to generate clusters (or to representative sequences from clusters created in this way); with closed-reference OTU picking we have a pre-established database of clusters and only allow matches against those, and with open-reference OTO picking we allow alignments against established clusters from a database, and we allow clusters to be generate de-novo.

We are going to use the latter method, and will be using the GreenGenes database as our pre-established reference.

The default procedure of picking OTU's will only match reads in their forward orientation against either the pre-established clusters or any new clusters. We want to make sure that our sequences can match in the forward and reverse orientation. To change the default behavior, we will use a parameter file to enforce both forward and reverse alignments.

```
pick_open_reference_otus.py -o otus/ -i combined/combined_seqs.fna -
p ./params.txt
```

[This step can take up to 15 minutes on this small data set]

Once this command finishes, copy the newly created 'otus' folder to your laptop. Navigate to the folder, and double-click the index.html file which will give us a quick overview of the files that have been generated. For most purposes, the BIOM (the Genomics Standards Consortium Biological Observation Matrix standard) formatted file otu_table_mc2_w_tax_no_pynast_failures.biom is the one we want to work with. All clusters that had only one sequence have been removed, as have clusters for which the representative sequence could not be aligned back to Greengenes.

The biom file is a binary file and not suitable for normal viewing with a text editor or with, e.g. the less command. Instead, we can use the biom command which is integrated into QIIME to have a closer look.

```
biom summarize-table -i
otus/otu_table_mc2_w_tax_no_pynast_failures.biom
```

Several of the remaining steps are based off the premise that there is an equal number of reads (~sequencing depth) between samples to do meaningful statistical testing. Of course, this is almost never the case, so we need to pick a depth where we keep the majority of the samples without losing too many sequences. In the output, check the counts/sample data. If we pick the median sampling depth, we'll be throwing away all samples with fewer than that depth, while at the same time the samples with more reads will be subsampled down to the median.

This is a difficult decision, and the answer will vary by experiment. For the tutorial we only have a few samples, and we want to keep all samples and conditions for the downstream analyses. We will pick the minimum sequencing depth (human: 3000, sorghum: 1675), so we keep all samples (although subsampled down).

4.2 Diversity analysis

```
core_diversity_analyses.py -o core_diversity/ -i
otus/otu_table_mc2_w_tax_no_pynast_failures.biom -m map.txt -t
otus/rep_set.tre -e XXXX
```

(replace the expected depth –e XXXXX by 3000 for human, and 1675 for sorgum).

(you can run ./scripts/core_human.sh or ./scripts/core_sorghum.sh instead)

This command will throw out some warnings – in this case we can ignore them.

Copy the 'core_diversity' folder to your laptop via Globus, and then open the index.html file. The index.html will have links to the output of the analysis as text files, binary biom files, and as plots.

Let's have a look at the bar plots first: they will give a high-level overview of the counts, per taxonomic unit, for each sample. The first bar plot deals with Phylum, the second with Class, then Order, etc. all the way down to Genus level. The area plots give a similar picture in a slightly different format.

The PCoA graph, both weighted and unweighted, show how similar (or not) different samples are. The initial view is not so interesting (all samples have the same color). Go to the colors tab and pick a metadata point to color the samples by[2].

During the diversity analysis all samples were treated individually, and hence some of the results (groupings) we see in the graphs might not we what we expected. We can run the diversity analysis again, but summarize the data over their metadata characteristics (e.g. group by nitrogen content, or diet).

Let us run the diversity analysis again, but with two sets of categorical data. For the sorghum data we'll use nitrogen level and genotype, for the human gut data we'll use diet and time point.

For the human data, enter:

```
core_diversity_analyses.py -o core_diversity/ --recover_from_failure –
c "Diet,Timepoint" -i otus/otu_table_mc2_w_tax_no_pynast_failures.biom
–m map.txt -t otus/rep_set.tre –e 3000
```

and for Sorghum:

```
core_diversity_analyses.py -o core_diversity/ --recover_from_failure –
c "Genotype,NitrogenLevel" -i
otus/otu_table_mc2_w_tax_no_pynast_failures.biom -m map.txt -t
otus/rep_set.tre –e 1675
```

[2] Alpha- and beta-diversity are measurements of diversity (species composition) within local sites, and between such sites respectively. See Whittaker, R. H. (1972). Evolution and Measurement of Species Diversity. Taxon, 21, 213-251. doi:10.2307/1218190

(you can use ./scripts/core_human_again.sh or ./scripts/core_sorghum_again.sh)

(the –recover_from_failure option has a two-fold purpose: it can help us pick up if your analysis stops halfway – for example when you get disconnected – and we can use It to skip some expensive initial calculations if we want to rerun with a different set of categories).

Then, again copy the 'core_diversity' folder to your desktop and determine what differences with the previous step you can find now (you might want to make a new folder first on your desktop, and copy the new results there so you can compare).

Fin.