A HTS Pipeline using USEARCH, Qiime, and FAST

Zewei Song

# Introduction

This is a pipeline for handling fungal ITS amplicon sequencing data. My general purpose is to create a pipeline to handle a single run of MiSeq in a personal laptop. As far as I tested, a full MiSeq run can be processed to generate a rarefied OTU table within several hours. A new script package - FAST - is used in this pipeline to connect varies programs. It will add a new label to the raw file, do quality filer, and generate a dereplicated OTU map and FASTA file that you can feed into varies OTU clustering program (including USEARCH 8.0 and anything in Qiime). It can also merge two OTU maps so you can do chain OTU clustering. Finally, I have a script to evaluate the quality of your mock community, and perform rarefaction with repeat subsampling on every sample.

This pipeline should also works for all types of amplicons (such as 18S, 16S), but I haven’t test on them yet.

# SECTION 0. SET UP WORKING ENVIRONMENT

The simplest way is to copy everything to your working folder. I think this is what you should start with if you are not familiar with the file system of your OS. For this pipeline, copy the USEARCH executable, .jar and .fa file of Trimmomatic, and all scripts and folder in FAST package to a new folder.

## 0.1 DOWLLOAD PROGRAMS

***USEARCH***

USEARCH: <http://www.drive5.com/usearch/download.html>

Submit your email address, and the website will send a link to your mailbox. Beware to choose the right platform for you. USEARCH will be a single file, with a version number. I always rename it to “usearch” for convenient.

***Fungal Amplicon Sequencing Toolbox***

I made up this name, let me know if you have better idea.

FAST: <https://github.com/ZeweiSong/FAST>

This is a GitHub repository with several script I wrote to pre-process the sequencing files, dereplicate, and handling the Qiime style OTU map. You can download the entire repository using “Download ZIP” button on the right and unzip the content to the same folder as USEARCH executable. You need to install Python 2.7 in order to use these scripts (<https://www.python.org/>). Python 3 should also work without any problem, but I haven’t test it under Python 3.

***Trimmomatic***

Trimmomatic: <http://www.usadellab.org/cms/index.php?page=trimmomatic>

Trimmomatic has to be in your working folder. You also need to install latest Java to make it work.

Also, create a new FASTA file named “illumina\_MiSeq\_Fungi.fa” and put in the sequences of your read1 and read2 primers (including reverse compliment). It should looks like:

>Read1

TTGGTCATTTAGAGGAAGTAAAAGTCGTAACAAGGTTTCC

>Read1\_rc

GGAAACCTTGTTACGACTTTTACTTCCTCTAAATGACCAA

>Read2

CGTTCTTCATCGATGCVAGARCCAAGAGATC

>Read2\_rc

GATCTCTTGGNTCTNGCATCGATGAAGAACG

## 0.2 SET UP PATH VARIABLE

Skip this part if you are not familiar with the file system of your OS.

In general, you can put the USEARCH executable and all FAST script in a single folder, and set the path as a path environment variable. By doing these, we can call these program anywhere in our computer.

For Windows user, add the path of the tool folder to the Path Variable of under User Variable. Following this to find the Path variable: Control Panel\System and Security\System\Advanced system setting\Environmental variables\User variables for User\_name

For Linux user, modify you .bashrc to include the folder path. Usually you can find this file under your home directory (~). Use vi ~/.bashrc to make any modification. Usually you would like to save the tool folder to /usr/local/bin.`

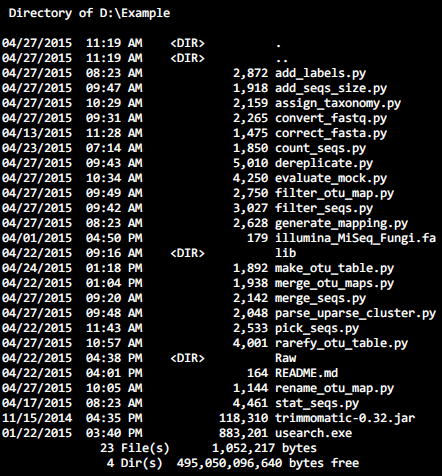
Use echo $PATH to check you current path variable.

## 0.3 SET UP WORKING FOLDER

Make a new folder, and under it make a new folder names “Raw”. Save all your R1 read FASTQ files to the “Raw” folder.

If you did not set up the path variable for the programs, copy USEARCH executable, and all FAST scripts including any subfolders to your working folder.

This is any example of the file structure you should have:



## 0.4 TEST RUN

For both Windows and Linux, use tab will autocomplete any file or folder in your current directory.

***All Python Scripts:***

@ python add\_labels.py -h

Use -h any time to see the usage for the scripts from FAST.

You should be able to run all python scripts by just typing its file name without starting with the word python. But if it doesn’t work, just start with “python”.

If you have Qiime installed, one script (merge\_otu\_maps.py) will have same name in FAST and Qiime. Both scripts will do the same thing, but if you copy all scripts to a working folder, the one from FAST will be called.

***Usearch:***

@ usearch

***Trimmomatic:***

@java -jar Trimmomatic-0.32.jar

# SECTION 1. LABEL SEQUENCE FILES

## 1.1 RELABEL FASTQ FILES:

***Generate File Mapping:***

@ generate\_mapping.py -i Raw -o mapping.txt

Open the mapping file and change any labels, the script will take a good guess basing on the file name.

***Add Labels:***

@ add\_labels.py -i Raw -o Labeled -m mapping.txt -t 4

Take a look at the files in the new folder “Labeled” using more under Windows, and less under Linux.

It should look like:

@SampleA\_1;SampleName=SampleA-1-R1;barcodelabel=SampleA;

GTAGGTGAACCTGCGGAAGGATCATTATCGATTCACGGGAAAGGGATGTGCTGGCGGAT...

+

CCCCCFFFFFCFGGGGGGFGGGHFHHHFHHGHHGHHHGGGGHHHGGHGHHHHHHGGGGF...

It is still a long name compared to the raw data. The part before the first “;”, SamnpleA\_1, is labels for Qiime, and the part after the first “;” is for USEARCH. They will not have conflict if you follow this pipeline, but you can also use -label to specify only one style of labels.

## 1.2 Merge FASTA Files:

***Merge All FASTQ File into One:***

@merge\_seqs.py -i Raw -o merged.fastq

***Count the Number of Total Sequences:***

@count\_seqs.py -i merged.fastq

# SECTION 2. Quality Filter

## 2.1 Quality Trim

@ java -jar trimmomatic-0.32.jar SE merged.fastq merged.trim.fastq ILLUMINACLIP:illumina\_MiSeq\_Fungi.fa:2:30:10 SLIDINGWINDOW:5:20 MINLEN:125

## 2.2 Ambiguous Base and Homopolyers

@ filter\_seqs.py -i merged.trim.fastq -o merged.trim.N0.homop9.fastq

## 2.3 Report on the Sequence File

@ stat\_seqs.py -i merged.trim.N0.homop9.fastq -o report.txt

## 2.4 Convert to FASTA File

@ convert\_fastq.py -i merged.trim.N0.homop9.fastq -o raw.qc.fasta

# SECTION 3. OTU CLUSTERING

## 3.1 Dereplication

@ dereplicate.py -i raw.qc.fasta -o raw.qc.derep -t 4

In general, USEARCH 8.0 has a faster dereplication function, but its free version has memory limitations (2 GB in Windows, 4 GB in Linux), and it does not output a Qiime style OTU map. A 2 GB file should be dereplicated in around 10 min using the script above. You should refrain from using other program since a large chunk of memory will be needed.

You will find two new files after dereplication:

raw.qc.derep.fasta contains only the unique sequences in raw.qc.fasta (like unique.seqs in mother).

raw.qc.derep.txt contains an OTU map that is compatible with Qiime.

## 3.2 Remove Singletons

@ filter\_otu\_map.py -i raw.qc.derep.txt -o raw.qc.derep.size2.txt -min\_size 2

@ pick\_seqs.py -i raw.qc.derep.fasta -o raw.qc.derep.size2.fasta -map raw.qc.derep.size2.txt

## 3.3 Cluster OTUs

***Add Size Annotation for USEARCH***

@ add\_seqs\_size.py -i raw.qc.derep.size2.fasta -map raw.qc.derep.size2.txt -o raw.qc.derep.size2.sizeout.fasta

***Cluster with USEARCH***

@ usearch -cluster\_otus raw.qc.derep.size2.fasta -uparseout uparse.up

***Convert Uparse Output to OTU Map***

@ parse\_uparse\_cluster.py -i uparse.up -o raw.qc.derep.size2.usearch.txt

***Merged Two OTU Maps***

@ merge\_otu\_maps.py -map\_large raw.qc.derep.size2.txt -map\_small raw.qc.derep.size2.usearch.txt -o merged.otu.txt

In general, any clustering method in Qiime can also be used here, as long as you remember to merge two OTU maps each time after clustering.

***Rename OTUs***

@ rename\_otu\_map.py -i merged.otu.txt -o merged.otu.renamed.txt

## 3.4 Make OTU Table

@ make\_otu\_table.py -i merged.otu.txt -o otu\_table.txt

## 3.5 Representative Sequence

Using Qiime (you need a Qiime virtual box under Windows: <http://qiime.org/install/virtual_box.html>)

@ pick\_rep\_seq.py -i merged.otu.txt -f raw.qc.fasta

The representative sequence will have a suffix “\_rep\_set”, so in this example it will be “raw.qc.fasta\_rep\_set.fasta”.

# SECTION 4 OTU TABLE

## 3.1 BLAST search representative sequences against UNITE database

***Make BLAST Database***

@ makeblastdb -in sh\_general\_release\_s\_02.03.2015.fasta -dbtype nucl -out unite\_02.03.2015

I would prefer to save the database file in a separated folder such as blast.

***BLAST***

@ blastn -db blast\unite\_02.03.2015 -query raw.qc.fasta\_rep\_set.fasta -max\_target\_seqs 1 -outfmt "6 qseqid stitle qlen length pident evalue" -out rep.otu.txt

You can choose to only output taxonomy information without any other output, but stick to this format if you want to use the following scripts.

## 3.2 Map BLAST result to the OTU table

@ assign\_taxonomy.py -otu otu\_table.txt -tax rep.otu.txt -o otu\_table.tax.txt

## 3.3 Filter low match records

I haven’t come up with a script here (will do in the future), but for now it is easy and flexible to do this in EXCEL. Filter out any rows that below a threshold of percentage match length (Subject\_Len/Query\_Len), and/or below a threshold of Pident.

## 3.4 Check the quality of mock community

BLAST on mock sequences database

@ makeblastdb -in mock\_sequences.fasta -dbtype nucl -out mock

@ blastn -db blast\mock -query raw.qc.fasta\_rep\_set.fasta -max\_target\_seqs 1 -outfmt "6 qseqid stitle qlen length pident evalue" -out mock\_blast.txt

***Generate a report on mock community***

@ assign\_taxonomy.py -otu otu\_table.txt -tax mock\_blast.txt -o otu\_table.mock.txt

@ evaluate\_mock.py -otu mock\_blast.txt -mock\_list mock\_list.txt -mock\_column Mock-community -o mock\_report.txt -min\_length 0.8 -min\_pident 85

mock\_list.txt is a file contains the Genus and species name (connected with “\_”, for example Fomes\_fomentarius, and Cortinarius\_sp.). The same name should appear in the mock sequence FASTA file using for making the BLAST database (here mock\_sequences.fasta).

Technically you can just use the OTU table with UNITE taxonomy here, but what I found out is that many mock OTUs were then binned to a “*Genus sp.”* instead of a “*Genus species”*, and thus cannot be identified. Since you can set the threshold for matched length and pident, low quality matches can be filtered out.

Take a look at the mock community report and determine if current OTU clustering is good enough. If you think another round of clustering is necessary, just remember to merge the new OTU map with the old one.

## 3.5 Subtract Negative Control

Subtract the abundance of negative control (if you have one) from other samples, so remove the OTU if you think it is necessary.

## 3.6 Rarefy OTU table

@rarefy\_otu\_table.py -otu otu\_table.tax.filtered.txt -o otu\_table.rare.txt -d 15000 -iter 1000 -thread 4 -keep\_all -meta\_column taxonomy

This does a similar but slightly better (I believe) job compared to single\_rarefaction.py in Qiime.

1) It can repeat the random subsample on every of your sample and pick one that has the medium richness. As far as I tested, if your sequencing depth is around 1/5 of the original total abundance, final richness will have a standard deviation around 10% of the original total abundance. I think pick the center point of a NMDS cloud is a better choice, but right now I don’t have an elegant solution for this.

2) It uses multithreads, in general 4 is a good number as I tested.

3) It kept all of your meta column if you specify the name of the first column of your meta data.

# SECTION 5 ANALYSIS

I’ll finish this part later, but both Qiime and R can do a decent job.