16S rRNA gene amplicon community analysis protocol

This protocol is modified based on the Mothur MiSeq Standard Operating Procedure.

This protocol was prepared using mothur version 1.38.1, R version 3.3.0, and phyloseq version 1.16.2.

1. Collect all gzipped FASTQ files into a single directory (your "project directory") - one pair of files (forward/reverse reads) for each sample.

2. Create a text file in the same folder as your `.fastq.gz` files to indicate to mothur which file matches which sample. The file should be called `basename.files` where `basename` is a short descriptive name for the dataset you're working with. The file should have one line per sample, with the following format (3-sample example):

   ```
   Sample1 Sample1_S20_L001_R1_001.fastq.gz    Sample1_S20_L001_R2_001.fastq.gz
   Sample2 Sample2_S19_L001_R1_001.fastq.gz    Sample2_S19_L001_R2_001.fastq.gz
   Sample3 Sample3_S21_L001_R1_001.fastq.gz    Sample3_S21_L001_R2_001.fastq.gz
   ```

   Optional: use the script fileListForMothur.sh to make this file

3. Open mothur and merge your forward and reverse reads and set the input/output directory (your "project directory") with the following command. The number of processors you choose will depend on your computer - if you don't know how many processors your computer has, then just set it to 1. Note that you will need to specify the drive and use backslashes for your directory paths on Windows (i.e. C:toproject_dir), the forward slashes shown below are for Mac or Linux.

   ```
   make.contigs(file=basename.files, inputdir=/path/to/project_directory, outputdir=/path/
   ```

4. Remove reads that are too short or suspiciously long. Here we're setting the allowable length range to 400-500 base pairs, but this will depend on the length of the amplicon that you're sequencing:

   ```
   screen.seqs(fasta=basename.trim.contigs.fasta, group=basename.contigs.groups, maxambig=
   ```

5. Find the unique sequences only to save time:

   ```
   unique.seqs(fasta=basename.trim.contigs.good.fasta)
   ```

6. So that mothur can keep track of the unique sequences across different samples, we need to run this command:

   ```
   count.seqs(name=basename.trim.contigs.good.names,group=basename.contigs.good.groups)
   ```

7. Align the sequences to a version of the silva database trimmed according to the primers you used for your sequencing. You can use the accompanying database preparation protocol to make your database. Re-

place `reference=/path/to/database/silva.nr_v123.pcr.align` in the following command with the path to your own database.

```
align.seqs(fasta=basename.trim.contigs.good.unique.fasta, reference=/path/to/database/s
```

8. Now run summary sequences on the aligned sequences to see how well they aligned.

```
summary.seqs(fasta=basename.trim.contigs.good.unique.align, count=basename.trim.contigs
```

9. Take a look at the summary sequences - the output will look something like this:

```
        Start   End NBases  Ambigs  Polymer NumSeqs
Minimum:    0   0   0   0   1   1
2.5%-tile:  6428    23440   64  0   4   3521
25%-tile:   6428    23440   403 0   4   35210
Median:     6428    23440   407 0   5   70420
75%-tile:   6428    23440   426 0   6   105630
97.5%-tile: 13859   23440   428 0   6   137319
Maximum:    26102   26103   447 0   12  140839
Mean:   7047.93 23349.2 395.411 0   4.89188
# of unique seqs:   99392
total # of seqs:    140839
```

10. The vast majority of sequences align from position 6428 to 23440, but a small minority of sequences do not. Use the following command to remove those sequences that are poorly aligned to the reference database. We also use this step to remove sequences with a string of identical bases (homopolymers) longer than 8 bases using the `maxhomop` command.

```
screen.seqs(fasta=basename.trim.contigs.good.unique.align, count=basename.trim.contigs.
```

11. Now if everything worked correctly your sequences should have aligned with actual base pairs in the database and not the flanking gaps (marked by a `.`). This command will filter out any bases that aligned outside the region of interest. It might be a good idea to run `summary.seqs()` again following this command to make sure you're not culling significant chunks of your alignment.

```
filter.seqs(fasta=basename.trim.contigs.good.unique.good.align, vertical=T, trump=.)
```

12. There is some redundancy across the sequences now following trimming, so we're going to run `unique.seqs` again.

```
unique.seqs(fasta=basename.trim.contigs.good.unique.good.filter.fasta, count=basename.t
```

13. Now to pre-cluster the sequences to remove sequencing error - this is thought to remove sequencing error (up to 1 bp variation per 100 bp). Since our amplicons are around 450bp long, we'll pre-cluster based on up to 4 differences.

```
pre.cluster(fasta=basename.trim.contigs.good.unique.good.filter.unique.fasta, count=bas
```

14. Now to detect and remove chimeric sequences using uchime.

```
chimera.uchime(fasta=basename.trim.contigs.good.unique.good.filter.unique.precluster.fa
remove.seqs(fasta=basename.trim.contigs.good.unique.good.filter.unique.precluster.fasta
```

15. Now the sequences are ready for classification.

```
classify.seqs(fasta=basename.trim.contigs.good.unique.good.filter.unique.precluster.pic
```

16. Now to cluster the sequences into OTUs using the vsearch agc (abundance-based greedy clustering) algorithm.

```
cluster(fasta=basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.fast
```

17. Determine the number of times each OTU occurs in each sample using this command.

```
make.shared(list=basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.a
```

18. Classify OTUs (based on the classification of the preclustered sequences) like so:

```
classify.otu(list=basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.
```

19. Get a file containing the consensus sequence for each OTU:

```
get.oturep(list=basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.ag
```

20. From this point you will switch from using mothur to using R. You can quit mothur using the `quit()` command, or leave it open if you think you might use it again. Open R and import your data using the phyloseq package with the following commands:

```
setwd("/path/to/project_directory")

library("phyloseq")
library("ggplot2")

moth_shared <- "basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.ag
moth_tax <- "basename.trim.contigs.good.unique.good.filter.unique.precluster.pick.agc.u
basename.phyloseq = import_mothur(mothur_shared_file=moth_shared, mothur_constaxonomy_f
```

21. If you sequenced a negative control sample, now would be a good time to check what's in there. You can get an overview of this with the following R command, "Negative_control" is the name you gave your negative control in the `basename.files` file you started out with in mothur. You should examine the negative control critically - are there any obvious contaminants? Remember that some of these OTUs may be spill over from your samples as a result of tag-switching. (Salter et. al. 2014) contains a good overview of some common contaminant genera found in reagents.

```
tax_table(basename.phyloseq)[row.names(otu_table(basename.phyloseq)[otu_table(basename.
```

22. Once you have some good candidate contaminant OTUs, you should check to see how abundant they are in your other samples. If they're a lot more abundant in your real samples, then they're probably in the negative control as a consequence of tag-switching. If they're a lot more abundant in your negative control, then it's probably a true contaminant. You can check the abundance of each candidate contaminating Otu with the following R code, assuming your candidate contaminants are Otu0075, Otu4471, and Otu0242:

```
candidate_contaminants <- c("Otu0075","Otu4471","Otu2042")
otu_table(basename.phyloseq)[candidate_contaminants,]
```

23. Once contaminating OTUs have been identified, you can remove them using the `prune_taxa` command within the R phyloseq package.

```
basename.phyloseq.decon <- prune_taxa(row.names(otu_table(basename.phyloseq)) %in% cand
```

24. If there are many contaminating OTUs within a single taxonomic lineage, it might make more sense to remove those OTUs based on their taxonomic classification rather than their OTU identifiers. This can be achieved in mothur using the `remove.lineage` command. An example to remove all *Pseudomonas* and *Propionibacter* is as follows, for more details see the remove.lineage command wiki page

```
remove.lineage(constaxonomy=basename.trim.contigs.good.unique.good.filter.unique.preclu
```

25. Plot alpha diversity estimates for all your samples:

```
plot_richness(basename.phyloseq)
```

26. Plot OTU abundance coloured by phylum-level classification:

```
plot_bar(basename.phyloseq, fill="Rank2")
```

27. Plot a basic NMDS (PCA) of your samples.

```
basename.phyloseq.ord <- ordinate(basename.phyloseq, "NMDS", "bray")
plot_ordination(basename.phyloseq, basename.phyloseq.ord, type="sample")
```

28. Phyloseq (and R, for that matter) contains many other powerful analysis and plotting tools for your data. See the Phyloseq Homepage for many good examples.