Analysis of Illumina MiSeq paired-end sequences of 16S rRNA amplicons
Brazelton Lab, November 2016
https://baas-becking.biology.utah.edu

This workflow was used for the following publications:
Dangerfield et al. (Submitted) Bacterial communities affected by experimental disturbances to canopy soils of a temperate rainforest. Appl Environ Microbiol.

Data processing primarily based on Schloss et al. (2013)
(http://www.mothur.org/wiki/MiSeq_SOP)
Other references:
Phyloseq (https://joey711.github.io/phyloseq/)
USEARCH (http://www.drive5.com/usearch/)

All Brazelton lab scripts are provide at our github site: https://github.com/Brazelton-Lab

Note that these commands are written exactly as implemented on the Brazelton lab server. Other users would have to adapt for their own system. For example, we use the cluster workload manager SLURM (http://slurm.schedmd.com/), which is why many commands begin with 'srun'.

Sequencing performed by Michigan State University
(https://rtsf.natsci.msu.edu/genomics/sequencing-services/ngs/illumina)

Description of how data is generated by MSU:
The V4 region of the 16S rRNA gene (515F/806R) was amplified using dual indexed Illumina fusion primers as described by Kozich, JJ et al[1]. After PCR the products were normalized and pooled using an Invitrogen SequalPrep DNA Normalization Plate. After library QC and quantitation the pool was loaded on an Illumina MiSeq v2 flow cell and sequenced using a standard 500 cycle reagent kit. Base calling was performed by Illumina Real Time Analysis (RTA) software v1.18.54. Output of RTA was demultiplexed and converted to FastQ files using Illumina Bcl2fastq v1.8.4.

[1] Kozich, J. J., Westcott, S. L., Baxter, N. T., Highlander, S. K., & Schloss, P. D. (2013). Development of a dual-index sequencing strategy and curation pipeline for analyzing amplicon sequence data on the MiSeq Illumina sequencing platform. Applied and Environmental Microbiology, 79(17), 5112–5120.http://doi.org/10.1128/AEM.01043-13

Keep a notebook: copy and paste all commands used for every step of this process into your personal notebook. Put the date and brief description of the project at the top.

Make a new directory for this project and link the appropriate files to it:

```
mkdir <project>
ln -svi /path/to/fastq_files/*  /path/to/new_project_directory/
cd <project>
```

Merge paired-end reads and filter the merged reads by quality scores using the max expected error method:
a. Create a slurm batch file with the following content (replace <> with appropriate information):

```
#! /bin/sh
#SBATCH -J <project>.preprocess
reads_path="</path/to/short_reads>";
files=$(find -L $reads_path -type f -regex ".*\.forward\.fastq.*");
for file in ${files[@]}; do
  file=$(basename $file);
  IFS=". " read -a array <<< $file;
  sample=${array[0]};
  forward=$reads_path"/"$sample".forward.fastq<.gz>";
  reverse=$reads_path"/"$sample".reverse.fastq<.gz>";
  vsearch --fastq_mergepairs $forward --reverse $reverse --log $sample.preprocess.log
--fastq_maxee 2 --fastaout $sample.merged.filtered.fasta;
done
```

b. Execute the batch script

```
sbatch <filename>
```

Note: this step will run vsearch, an open source alternative for usearch, on all fastq files in the specified directory. It assumes that files are named with sample name as the prefix, fastq or fastq.gz as the suffix, strands are differentiated with the terms forward and reverse, and that components of the file name are delimited with a period (e.g. 0RBCd001.forward.fastq.gz and 0RBCd001.reverse.fastq.gz). If a different naming convention was used, replace the regular expression (regex) in the find command with the appropriate string that identifies forward reads from reverse reads. In almost all cases, this should either be `".*_R1_.*"` (vs `".*_R2_.*"`) or `".*\.forward\..*"` (vs `".*\.reverse\..*"`). Also, make sure that the project directory contains only the fastq files that you want to process.
Additional Note: vsearch, unlike usearch, has support for compressed (i.e. gzipped) input. This means that there is no need to uncompress the raw fastq files prior to running this step.

Make a group file (tab-delimited file containing all of the sequence ids in a project with their associated sample):

```
srun group_from_filenames.py --separator . --position 1
*.merged.filtered.fasta > <project>.group
```

Combine all samples from the same project:

```
cat *.merged.filtered.fasta > <project>.merged.filtered.fasta
```

Now in mothur:

```
srun --ntasks <CPUs> --pty mothur
```

Note: some steps in mothur can take a long time to complete (align.seqs, chimera.uchime, and

classify.seqs primarily). It might be easier to run these steps individually as a batch job. This can be done by creating a batch script with your favorite text editor (ex: chimera_batch.sh) with the following content:

```
#! /bin/sh
#SBATCH --ntasks <CPUs>
mothur "#mothur_command(command_arguments, processors=<CPUs>)"
```

Remember that the use of "current" in place of the file name is disallowed outside of an interactive session of mothur. The full names of the files will need to be provided. To then run the batch script, use:

```
sbatch <batch_script>
```

```
count.groups(group=<project>.group)
unique.seqs(fasta=<project>.merged.filtered.fasta)
summary.seqs(fasta=current, name=current, processors=<CPUs>)
```

[copy and paste results from summary into your notebook]

```
screen.seqs(fasta=current, name=current, group=current, maxambig=0,
maxhomop=8)
count.seqs(name=current, group=current)
summary.seqs(count=current)
```

[copy and paste results from summary into your notebook]

Perform chimera checking using Mothur's implementation of Uchime:

```
chimera.uchime(fasta=current, count=current, dereplicate=t)
#  the query sequence is referenced against the more abundant sequences from the same
sample
remove.seqs(fasta=current, accnos=current, count=current, dups=f)
summary.seqs(fasta=current, count=current)
```

[copy and paste results from summary into your notebook]

Check what the most recent version of the reference database is

```
align.seqs(fasta=current,
reference=/srv/databases/markers/silva/silva.SSURef_123.1/silva123-1.nr99.ali
gn.trunc.bact.fasta)
summary.seqs(fasta=current, count=current)
```

[copy and paste results from summary into your notebook]

```
filter.seqs(fasta=current, vertical=T)
summary.seqs(fasta=current, count=current)
```

[copy and paste results from summary into your notebook]

```
unique.seqs(fasta=current, count=current)
summary.seqs(fasta=current, count=current)
```
[copy and paste results from summary into your notebook]

```
pre.cluster(fasta=current, count=current, diffs=1)
summary.seqs(fasta=current, count=current)
```
[copy and paste results from summary into your notebook]

```
classify.seqs(fasta=current, count=current,
reference=/srv/databases/markers/silva/silva.SSURef_123.1/silva123-1.nr99.ali
gn.trunc.bact.fasta,
taxonomy=/srv/databases/markers/silva/silva.SSURef_123.1/silva123-1.nr99.alig
n.trunc.bact.taxonomy, cutoff=80)
```

Now you can leave mothur:
```
quit()
```

compress or remove the alignment file:
```
gzip --best *.align
```
or (preferred)
```
rm *.align
```

Remove other unnecessary files:
```
rm *pick.count_table
rm *good.count_table
rm *filter.count_table
rm *filter.fasta
rm *unique.fasta
rm *.map
```

Output 1: The final count_table - the one with the longest name and ending with "precluster.count_table" - is one of the final products. Copy this to a different directory where you will collect all of the final output files.
For example:
mkdir final_output
cp *.precluster.count_table final_output/

Convert the count table into a shared file:
```
srun convert_count_to_shared.py <count_table_file>
```

Start mothur again to calculate some diversity statistics:
```
srun --ntasks <CPUs> --pty mothur
```

```
rarefaction.single(shared=<shared_file>)
summary.single(shared=current,
calc=sobs-chao-ace-shannon-simpson-invsimpson-simpsoneven)
summary.shared(shared=current,
calc=sharedsobs-sorclass-sorabund-braycurtis-morisitahorn)
tree.shared(shared=current, calc=morisitahorn)
tree.shared(shared=current, calc=sorclass)
heatmap.sim(shared=current, calc=morisitahorn-sorclass)
quit()
```

Delete the .rabund files:

```
rm *.rabund
```

Output 2: Copy the SIX resulting files into the final output directory. These files end with '.groups.rarefaction', '.groups.summary', '.count_table.summary', '.tre', and '.sim.svg

Add full phylogeny to taxon name and optionally split summary by taxonomic levels:

```
srun modify_tax_summary.py -l 3,4,5,6 <filename.tax.summary>
```

Note: The numbers do not directly correspond to a taxonomic level for all sequences but, generally, 3 = class, 4 = order, 5 = family, and 6 = genus.

Output 3: Copy these modified taxonomy summary files into the final output directory.

Generate a krona graph from the taxonomy summary file:

```
srun plot_tax_summary.py --split <filename.tax.summary> <output_prefix>
mkdir krona
mv *.krona krona/
srun ktImportText -o <prefix>.krona.html krona/*.krona
```

Output 4: Copy this krona.html file into the final output directory.

Generate taxonomy bar plot with phyloseq

Starting materials:

- count table from mothur
- taxonomy table from mothur

Prepare taxonomy table from mothur for use in phyloseq:

```
srun taxonomy2tsv.py <filename>.taxonomy
```

Create phyloseq otus-tax-sample object in R

```
srun PhyloSeq-Barplot.R --count_table <filename>.count_table --tsv
<filename>.taxonomy.tsv --merge_threshold 0.5 --file_type png --level 6
--output family.barplot
```

The “--level” option can be 2, 3, 4, 5, 6... If multiple different taxonomic levels are desired or requested by a collaborator, re-run this command once with each different level; make sure to change the filename of the output according to the taxonomic level used.

Merge the taxonomy file with the count table for ease of viewing the results:

```
srun count_cat_tax.py --tsv <filename>.taxonomy.tsv --count_table
<filename>.count_table --output <filename>.count_table.taxonomy.tsv
```

Output 5: Copy the .png file and the count_table.taxonomy.tsv file into the final output directory.

The End!

Description of Output Files

1. The file ending in **'.precluster.count_table'** is a tab-delimited text file that you could open with a text editor or with a spreadsheet program like Excel. It lists the number of times each sequence occurs in each sample.
2. The **'.groups.rarefaction'** file is a tab-delimited text file that you could open with a text editor or with a spreadsheet program like Excel. It contains the data you would need to make a rarefaction plot, which shows the number of different kinds of sequences in each sample as a function of the total number of sequences in that sample. The **'.groups.summary'** file is a tab-delimited text file that you could open with a text editor or with a spreadsheet program like Excel. It contains various alpha diversity statistics for each sample. Consult http://mothur.org/wiki/Calculators for more information. The **'.count_table.summary'** file is a tab-delimited text file that you could open with a text editor or with a spreadsheet program like Excel. It contains the number of shared sequences and the percent **dissimilarity** between each pair of samples according to three different beta diversity calculators (see above). The **'.tre'** files are dendrograms that can be opened in any phylogenetic tree software such as Dendroscope. They depict the whole-community similarities among the samples in the study according to binary presence/absence of shared sequences (sorclass) or according to community structure including differential abundance of shared sequences (morisitahorn). The **'sim.svg'** file is an image file that can be opened with Firefox or (preferably) with an SVG viewer like GIMP. It shows the same information as the **'.tre'** files, except that similarities are shown as gradations of color rather than as a hierarchical dendrogram. The sample labels are impossible to read. Sorry, I don't know how to solve this yet. You could make your own heat map with the information in the **'.count_table.summary'** file.
3. The **'.tax.summary.*'** files are tab-delimited text files that you could open with a text editor or with a spreadsheet program like Excel. They report the taxonomic classifications of each sequence and the abundance of that taxonomic classification in each sample. The '.tax.summary.mod' file includes all taxonomic ranks together, and the other files contain only one taxonomic rank.
4. The **'.krona.html'** file can be opened with any web browser. It provides a graphical

visualization of the taxonomical classifications in the '.tax.summary.*' files described above. The charts are interactive and can be explored by double-clicking on a section of the chart. Only one sample is shown at a time, and other samples can be chosen by clicking on the sample name in the menu on the left-hand side.

5. The **'count_table.taxonomy.tsv'** is the count_table described in output #1, amended with the taxonomic classifications from output #3. The **'.barplot.png'** file is an image file showing the taxonomic classifications for each sample. The bar plot for each sample orders (from left to right) the taxonomic classifications from most abundant to least abundant. See the help documentation for PhyloSeq-Barplot.R to read how this script can customize the appearance of the bar plot.