This exercise is adapted from the Mothur MiSeq SOP
http://www.mothur.org/wiki/MiSeq_SOP

Kozich JJ, Westcott SL, Baxter NT, Highlander SK, Schloss PD. (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):5112-20.

Reducing sequencing and PCR errors

In this exercise we are going to examine the microbiome from the pig intestinal track from one donor. These sequences were generated by 454 downloaded from SRA.

Looft T, Allen HK, Cantarel BL, Levine UY, Bayles DO, Alt DP, Henrissat B, Stanton TB. Bacteria, phages and pigs: the effects of in-feed antibiotics on the microbiome at different gut locations. ISME J. 2014 Aug;8(8):1566-76. doi: 10.1038/ismej.2014.12. Epub 2014 Feb 13. PubMed PMID: 24522263.

First lets examine the sequence data

```
summary.seqs(fasta=swine_gut.fasta)
```

Now lets trim sequences based on quality, remove any large, small, ambiguious or reads with lots of homopolymers

```
trim.seqs(fasta=swine_gut.fasta,qfile=swine_gut.qual,qaverage=25,maxambig=0,maxhomop=8,minlength=200,maxlength=800,processors=2)
```

Now we are going to reduce the number of reads for analysis. Don't worry we are "keeping" these sequences for abundance purposes, just not for analysis.

```
unique.seqs(fasta=swine_gut.trim.fasta)
```

Processing improved sequences

Normally, we can limit the alignment to the variable region of the 16S gene based on the primers. First thing we would do is to reduce the 16S reference alignment to this region. But this step is very long, so we are going to use the precut alignment.

```
#pcr.seqs(fasta=silva.nr_v119.align, start=1044, end=6500,keepdots=F)
```

Next align sequences to template 16S rRNA gene. We can remove any sequences that don't align to the expected PCR region. The alignment step can also be very long – so you should probably skip this step too.

```
#align.seqs(fasta=swine_gut.trim.unique.fasta,template=silv
a.nr_v119.pcr.align,flip=t,processors=2)
```

Summarize results so far

```
summary.seqs(fasta=swine_gut.trim.unique.align,
name=swine_gut.trim.names)
```

Next we want to create a file to keep track of the count of our representative sequences in each sample. Since we skipped some the demultiplexing steps, well have to generate the read to same file (group file). Then create a count table.

```
system(perl make_group_names.pl swine_gut.trim.names)
count.seqs(name=swine_gut.trim.names,
group=swine_gut.trim.groups)
```

Now we want to "clean" up our alignment, determine the sequences that cover most of the PCR region, cut the alignment to that region and remove those that don't span the whole region

```
screen.seqs(fasta=swine_gut.trim.unique.align,
count=swine_gut.trim.count_table,start=2,optimize=end,
criteria=85, processors=2)
filter.seqs(fasta=swine_gut.trim.unique.good.align,
vertical=T, trump=., processors=2)
```

Now that our alignment is of high quality, let's remove redundant sequences and cluster sequences with only 2 mismatches (these could represent sequencing errors)

```
unique.seqs(fasta=swine_gut.trim.unique.good.filter.fasta,
count=swine_gut.trim.good.count_table)
pre.cluster(fasta=swine_gut.trim.unique.good.filter.unique.
fasta,count=swine_gut.trim.unique.good.filter.count_table,d
iffs=2)
```

We want to identify and remove chimera sequences. This step is really slow, we have already identified these sequences. So skip this step and remove the chimeras.

```
#chimera.uchime(fasta=swine_gut.trim.unique.good.filter.uniq
ue.precluster.fasta,
reference=silva.gold.align,processors=3)
remove.seqs(accnos=swine_gut.trim.unique.good.filter.unique
.precluster.ref.uchime.accnos,
fasta=swine_gut.trim.unique.good.filter.unique.precluster.f
asta,count=swine_gut.trim.unique.good.filter.unique.preclus
ter.count_table)
```

Taxonomic Classification

#Classify Sequences into Taxonomic Bins

```
classify.seqs(fasta=swine_gut.trim.unique.good.filter.uniqu
e.precluster.pick.fasta, template=silva.nr_v119.pcr.align,
taxonomy=silva.nr_v119.tax, cutoff=50, processors=2)
```

This command allows you to remove suspicious sequences, for example if you susptect Cyanobacteria as representing chloroplast from plant material (which it could in the gut), you can remove it

```
remove.lineage(fasta=swine_gut.trim.unique.good.filter.uniq
ue.precluster.pick.fasta,
count=swine_gut.trim.unique.good.filter.unique.precluster.p
ick.count_table,
taxonomy=swine_gut.trim.unique.good.filter.unique.precluste
r.pick.nr_v119.wang.taxonomy,taxon=Cyanobacteria)
```

#mothur has a tendency to make long complicated names, these commands allows you to rename files to shorter names

```
system(cp
swine_gut.trim.unique.good.filter.unique.precluster.pick.pi
ck.fasta final.fasta)
system(cp
swine_gut.trim.unique.good.filter.unique.precluster.pick.pi
ck.count_table final.count_table)
system(cp
swine_gut.trim.unique.good.filter.unique.precluster.pick.nr
_v119.wang.pick.taxonomy final.taxonomy)
```

Preparing for OTU analysis

Create a distance matrix comparing 16S sequences in your samples. Then cluster sequences into OTU -- we are using the command cluster.split to do that as it allows us to cluster sequences according a taxonomic level like order or family

```
dist.seqs(fasta=final.fasta,cutoff=0.15, processors=2)
cluster.split(column=final.dist, taxonomy=final.taxonomy,
count=final.count_table, splitmethod=classify, taxlevel=4,
processors=1)
```

To know the taxonomy for each of our OTUs, get the consensus taxonomy for each OTU

```
classify.otu(list=final.an.unique_list.list,
count=final.count_table, taxonomy=final.taxonomy,
label=0.03)
```

#Calculate the relative abundance of taxa

```
phylotype(taxonomy=final.taxonomy,count=final.count_table)
```

The number of sequences in your count table should match the number in your OTU list file. If time permits create a PERL script to removes these extra sequences from the count file -- call this new file final.fix.count_table. *Otherwise use fix_ct.pl*

Analysis

#Generates collector's curves
#The make.shared creates a file that represent the number of times that an OTU is observed in multiple samples

```
make.shared(list=final.an.unique_list.list,count=final.fix.
count_table, label=0.03)
```

Let's start our analysis by analyzing the alpha diversity of the samples. First we will generate collector's curve of the Chao1 richness estimators and the inverse Simpson diversity index.

```
collect.single(shared=final.an.unique_list.shared,
calc=chao-invsimpson, freq=100)
```

This command will generate file ending in *.chao and *.invsimpson for each sample, which can be plotted in your favorite graphing software package. It is important to point out that Chao1 is really a measure of the minimum richness in a community, not the full richness of the community. Also, there isn't much to do with these plots since it's based on a single sampling. One method often used to get around this problem is to look at rarefaction curves describing the number of OTUs observed as a function of sampling effort.

```
rarefaction.single(shared=final.an.unique_list.shared,
calc=sobs, freq=100)
```

#Determine the sequence count per sample

```
count.groups(shared=final.an.unique_list.shared)
```

This will generate files ending in *.rarefaction, which again can be plotted in your favorite graphing software package. Alas, rarefaction is not a measure of richness, but a measure of diversity. If you consider two communities with the same richness, but different evenness then after sampling a large number of individuals their rarefaction curves will asymptote to the same value. Since they have different evennesses the shapes of the curves will differ. Therefore, selecting a number of individuals to cutoff the rarefaction curve isn't allowing a researcher to compare samples based on richness, but their diversity. Finally, let's get a table containing the number of sequences, the sample coverage, the number of observed OTUs, and the Inverse Simpson diversity estimate using the summary.single command. To standardize everything, let's randomly select sequences from each sample 1000 times and calculate the average (note: that if we set subsample=T, then it would use the size of the smallest library):

```
summary.single(shared=final.an.unique_list.shared,
calc=nseqs-coverage-sobs-invsimpson, subsample=2725)
```

Subsample the OTUs using the minimum number of sequences. Then, let's determine whether our data can be partintioned in to separate community types.

```
sub.sample(shared=final.an.unique_list.shared, size=2725)
get.communitytype(shared=final.an.unique_list.0.03.subsampl
e.shared)
```

The unifrac-based metrics are used to assess the similarity between two communities membership (unifrac.unweighted) and structure (unifrac.weighted). We will use these metrics and generate PCoA plots to compare our samples. There are two beta-diversity metrics that one can use - unweighted and weighted. We will also have mothur subsample the trees 1000 times and report the average. In order to do unifract analysis (beta diversity), we need to determine the distances between sequences, build a phylogentic tree and then

compare samples using unifract. Principal Coordinates (PCoA) uses an eigenvector-based approach to represent multidimensional data in as few dimesnsions as possible. Our data is highly dimensional (~9 dimensions). Tree building is a very slow step. Please use the pre-generated tree.

```
#dist.seqs(fasta=final.fasta,cutoff=0.15,
output=lt,processors=4)
#clearcut(phylip=final.phylip.dist)
unifrac.unweighted(tree=final.phylip.tre,
count=final.count_table, distance=lt, processors=2,
random=F, subsample=2725)
unifrac.weighted(tree=final.phylip.tre,
count=final.count_table, distance=lt, processors=2,
random=F, subsample=2725)
pcoa(phylip=final.phylip.1.unweighted.ave.dist)
pcoa(phylip=final.phylip.tre1.weighted.ave.dist)
```

We can do the same analysis using the similarity of the membership and structure found in the various samples

```
dist.shared(shared=final.an.unique_list.shared,
calc=thetayc-jclass, subsample=2725)
pcoa(phylip=final.an.unique_list.thetayc.0.03.lt.dist)
nmds(phylip=final.an.unique_list.thetayc.0.03.lt.dist)
```

Finally, no microbiome study would be complete without a few heatmaps. We can visualize those distances as similarities in a heatmap with the Jaccard and thetayc coefficients. We can also visualize relative abundance of each OTU across the 24 samples using the heatmap.bin command and log2 scaling the relative abundance values. Because there are so many OTUs, let's just look at the top 50 OTUs.

```
heatmap.bin(shared=final.an.unique_list.shared, scale=log2,
numotu=50)
heatmap.sim(phylip=final.an.unique_list.thetayc.0.03.lt.dis
t)
heatmap.sim(phylip=final.an.unique_list.jclass.0.03.lt.dist
)
```