MiSeq SOP

From mothur

NOTE: Although this is an SOP, it is something of a work in progress and continues to be modified as we learn more. If you are using this protocol in a paper, you must cite the Schloss et al. 2013 AEM paper and cite the date you accessed this page:

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.

The goal of this tutorial is to demonstrate the standard operating procedure (SOP) that the Schloss lab uses to process their 16S rRNA gene sequences that are generated using Illumina's MiSeq platform using paired end reads. The approach we take is to use index reads to multiplex a large number of samples (i.e. 384) on a single run. You can also see our latest wet-lab SOP (https://github.com/SchlossLab/MiSeq_WetLab_SOP) for generating these libraries. Others have generated similar data but without the index reads and so the index (aka barcode) sequences are found at the beginning of each read. This SOP will highlight the differences in processing between these two approaches. This SOP is largely the product of a series of manuscripts that we have published and users are advised to consult these for more details and background data. The MiSeq-specific steps are described in a manuscript that is in review with Applied and Environmental Microbiology. The workflow is being divided into several parts shown here in the table of contents for the tutorial:

- Korean translation of MiSeq_SOP (http://metagenomics.tistory.com/entry/Mothur-파이프라인-pipeline)

Contents

Logistics

Starting out we need to first determine, what is our question? The Schloss lab is interested in understanding the effect of normal variation in the gut microbiome on host health. To that end we collected fresh feces from mice on a daily basis for 365 days post weaning (we're accepting applications). During the first 150 days post weaning (dpw), nothing was done to our mice except allow them to eat, get fat, and be merry. We were curious whether the rapid change in weight observed during the first 10 dpw affected the stability microbiome compared to the microbiome observed between days 140 and 150. We will address this question in this tutorial using a combination of OTU, phylotype, and phylogenetic methods. To make this tutorial easier to execute, we are providing only part of the data - you are given the flow files for one animal at 10 time points (5 early and 5 late). In addition, to sequencing samples from mice fecal material, we resequenced a mock community composed of genomic DNA from 21 bacterial strains. We will use the 10 fecal samples to look at how to analyze microbial communities and the mock community to measure the error rate and its effect on other analyses.

In a manuscript submitted to Applied & Environmental Microbiology, we describe a set of primers that will allow you to sequence 1536 samples in parallel using only 80 primers (32+48) and obtain sequence reads that are at least as good as those generated by 454 sequencing using our 454 SOP. Please consult the supplementary methods of that manuscript for more information and our wet-lab SOP. All of the data from that study are available through our server. Sequences come off the MiSeq as pairs of fastq files with each pair representing the two sets of reads per sample. fastq files contain both the sequence data and the quality score data. If you aren't getting these files off the sequencer, then you likely have the software parameters set incorrectly. For this tutorial you will need several sets of files. To speed up the tutorial we provide some of the downstream files that take awhile to generate (e.g. the output of shhh.flows):

- Example data from Schloss lab that will be used with this tutorial. It was extracted from the full dataset (http://www.mothur.org/MiSeqDevelopmentData/StabilityNoMetaG.tar)
- SILVA-based bacterial reference alignment
- mothur-formatted version of the RDP training set (v.9)

You can easily substitute these choices (and should) for the reference and taxonomy alignments using the updated Silva reference files, RDP reference files, and Greengenes-formatted databases. We use the above files because they're compact and do a pretty good job. The various classification references perform differently with different sample types so your mileage may vary. It is generally easiest to decompress these files and to then move the contents of the Trainset9_032012.pds and the silva.bacteria folders into the MiSeq_SOP folder. You will also want to move the contents of the mothur executable folder there as well. If you are a sysadmin wiz (or novice) you can probably figure out how to put mothur in your path, but this will get you what you need for now.

In addition, you probably want to get your hands on the following...

- mono (http://www.mono-project.com/Main_Page) - if you are using Mac OS X or linux
- TextWranger (http://www.barebones.com/products/textwrangler/) / emacs (http://www.gnu.org/software/emacs/) / vi (http://www.vim.org/) / or some other text editor
- R (http://www.r-project.org/), Excel, or another program to graph data
- Adobe Illustrator, Safari, or Inkscape (http://inkscape.org/)
- TreeView (http://taxonomy.zoology.gla.ac.uk/rod/treeview.html), FigTree (http://tree.bio.ed.ac.uk/software/figtree/) or another program to visualize dendrograms

It is generally easiest to use the "current" option for many of the commands since the file names get very long. Because this tutorial is meant to show people how to use mothur at a very nuts and bolts level, we will only selectively use the current option to demonstrate how it works. Generally, we will use the full file names for this tutorial.

Getting started

Because of the large size of the original dataset (3.9 GB) we are giving you 21 of the 362 pairs of fastq files. For example, you will see two files: F3D0_S188_L001_R1_001.fastq and F3D0_S188_L001_R2_001.fastq. These two files correspond to Female 3 on Day 0 (i.e. the day of weaning). The first and all those with R1 correspond to read 1 while the second and all those with R2 correspond to the second or reverse read. These sequences are 250 bp and overlap in the V4 region of the 16S rRNA gene; this region is about 253 bp long. So looking at the files in the MiSeq_SOP folder that you've downloaded you will see 22 fastq files representing 10 time points from Female 3 and 1 mock community. You will also see HMP_MOCK.v35.fasta which contains the sequences used in the mock community that we sequenced in fasta format. Finally you will see a file called stability.files. The first lines of this file look like:

```
F3D0    F3D0_S188_L001_R1_001.fastq     F3D0_S188_L001_R2_001.fastq
F3D141  F3D141_S207_L001_R1_001.fastq   F3D141_S207_L001_R2_001.fastq
F3D142  F3D142_S208_L001_R1_001.fastq   F3D142_S208_L001_R2_001.fastq
F3D143  F3D143_S209_L001_R1_001.fastq   F3D143_S209_L001_R2_001.fastq
```

```
F3D144  F3D144_S210_L001_R1_001.fastq   F3D144_S210_L001_R2_001.fastq
...
```

Mothur can create this file for you using the make.file command.

```
mothur > make.file(inputdir=., type=fastq, prefix=stability)
```

The first column is the name of the sample. The second column is the name of the forward read for that sample and the third columns in the name of the reverse read for that sample. Pretty easy, eh? Finally, there's a batch file included that we'll discuss at the end of the tutorial.

Reducing sequencing and PCR errors

The first thing we want to do is combine our two sets of reads for each sample and then to combine the data from all of the samples. This is done using the make.contigs command, which requires stability.files as input. This command will extract the sequence and quality score data from your fastq files, create the reverse complement of the reverse read and then join the reads into contigs. We have a very simple algorithm to do this. First, we align the pairs of sequences. Next, we look across the alignment and identify any positions where the two reads disagree. If one sequence has a base and the other has a gap, the quality score of the base must be over 25 to be considered real. If both sequences have a base at that position, then we require one of the bases to have a quality score 6 or more points better than the other. If it is less than 6 points better, then we set the consensus base to an N. Let's give it a shot (I'm using 8 processors, because my computer has 8 processors, use what you've got)...

```
mothur > make.contigs(file=stability.files, processors=8)
```

The first thing you'll see is that it processes the fastq files to generate the individual fasta and qual files. Then it will go through each set of files and make the contigs. This took about 84 seconds on my computer. Clearly, it will take longer for a full dataset. In the end it will tell you the number of sequences in each sample:

```
Group count:
F3D0    7793
F3D1    5869
F3D141  5958
F3D142  3183
F3D143  3178
F3D144  4827
F3D145  7377
F3D146  5021
F3D147  17070
F3D148  12405
F3D149  13083
F3D150  5509
F3D2    19620
F3D3    6758
F3D5    4448
```

```
F3D6    7989
F3D7    5129
F3D8    5294
F3D9    7070
Mock    4779
Total of all groups is 152360
```

At the very end it will give you the following warning message:

```
[WARNING]: your sequence names contained ':'.  I changed them to '_' to avoid problems in your downstream analysis.
```

Don't worry too much about this. The typical sequence name will look like "M00967:43:000000000-A3JHG:1:1101:18327:1699". Aside being freakishly long, these sequence names contain ":", which will cause a lot of headaches down the road if you are crazy enough to try and create phylogenetic trees from these sequences. So to prevent this headache for you, we convert all of the ":" characters to "_" characters. This command will also produce several files that you will need down the road: stability.trim.contigs.fasta and stability.contigs.groups. These contain the sequence data and group identity for each sequence. The stability.contigs.report file will tell you something about the contig assembly for each read. Let's see what these sequences look like using the summary.seqs command:

```
mothur > summary.seqs(fasta=stability.trim.contigs.fasta)

                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1       248     248     0       3       1
2.5%-tile:      1       252     252     0       3       3810
25%-tile:       1       252     252     0       4       38091
Median:         1       252     252     0       4       76181
75%-tile:       1       253     253     0       5       114271
97.5%-tile:     1       253     253     6       6       148552
Maximum:        1       503     502     249     243     152360
Mean:           1       252.811 252.811 0.697867        4.44854
# of Seqs:      152360
```

This tells us that we have 152360 sequences that for the most part vary between 248 and 253 bases. Interestingly, the longest read in the dataset is 502 bp. Be suspicious of this. Recall that the reads are supposed to be 251 bp each. This read clearly didn't assemble well (or at all). Also, note that at least 2.5% of our sequences had some ambiguous base calls. We'll take care of these issues in the next step when we run screen.seqs.

```
mothur > screen.seqs(fasta=stability.trim.contigs.fasta, group=stability.contigs.groups, maxambig=0, maxlength=275)
```

This implementation of the command will remove any sequences with ambiguous bases and anything longer than 275 bp. There's another way to run this using the output from summary.seqs:

```
mothur > screen.seqs(fasta=stability.trim.contigs.fasta, group=stability.contigs.groups, summary=stability.trim.contigs.summary, maxambig=0, maxlength=275)
```

This may be faster because the summary.seqs output file already has the number of ambiguous bases and sequence length calculated for your sequences. Also, mothur is smart enough to remember that we used 8 processors in make.contigs and so it will use that throughout your current session. To see what else mothur knows about you, run the following:

```
mothur > get.current()

Current files saved by mothur:
fasta=stability.trim.contigs.good.fasta
group=stability.contigs.good.groups
processors=8
```

What this means is that mothur remembers your latest fasta file and group file as well as the number of processors you have. So you could run:

```
mothur > summary.seqs(fasta=stability.trim.contigs.good.fasta)
mothur > summary.seqs(fasta=current)
mothur > summary.seqs()
```

and you would get the same output for each. For the purposes of this tutorial we will write out the names of the files. At this point our sequencing error rate has probably dropped more than an order of magnitude and we have 128872 sequences. Let's press on...

Processing improved sequences

We anticipate that many of our sequences are duplicates of each other. Because it's computationally wasteful to align the same thing a bazillion times, we'll unique our sequences using the unique.seqs command:

```
mothur > unique.seqs(fasta=stability.trim.contigs.good.fasta)
```

If two sequences have the same identical sequence, then they're considered duplicates and will get merged. In the screen output there are two columns - the first is the number of sequences characterized and the second is the number of unique sequences remaining. So after running unique.seqs we have gone from 128872 to 16426 sequences. This will make our life much easier. Another thing to do to make our lives easier is to simplify the names and group files. If you look at the most recent versions of those files you'll see together they are 13 MB. This may not seem like much, but with a full MiSeq run those long sequence names can add up and make life tedious. So we'll run count.seqs to generate a table where the rows are the names of the unique sequences and the columns are the names of the groups. The table is then filled with the number of times each unique sequence shows up in each group.

```
mothur > count.seqs(name=stability.trim.contigs.good.names, group=stability.contigs.good.groups)
```

This will generate a file called stability.trim.contigs.good.count_table. In subsequent commands we'll use it by using the count option:

```
mothur > summary.seqs(count=stability.trim.contigs.good.count_table)

Using stability.trim.contigs.good.unique.fasta as input file for the fasta parameter.

Using 8 processors.

                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1       250     250     0       3       1
2.5%-tile:      1       252     252     0       3       3222
25%-tile:       1       252     252     0       4       32219
Median:         1       252     252     0       4       64437
75%-tile:       1       253     253     0       5       96655
97.5%-tile:     1       253     253     0       6       125651
Maximum:        1       270     270     0       12      128872
Mean:   1       252.462 252.462 0       4.36693
# of unique seqs:       16426
total # of seqs:        128872
```

Cool, right? Now we need to align our sequences to the reference alignment. Again we can make our lives a bit easier by making a database customized to our region of interest using the pcr.seqs command. To run this command you need to have the reference database (silva.bacteria.fasta) and know where in that alignment your sequences start and end. To remove the leading and trailing dots we will set keepdots to false. You could also run this command using your primers of interest.:

```
mothur > pcr.seqs(fasta=silva.bacteria.fasta, start=11894, end=25319, keepdots=F, processors=8)
```

Let's rename it to something more useful using the rename.file command:

```
mothur > rename.file(input=silva.bacteria.pcr.fasta, new=silva.v4.fasta)
```

Let's take a look at what we've made:

```
mothur > summary.seqs(fasta=silva.v4.fasta)
```

```
                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1       13424   270     0       3       1
2.5%-tile:      1       13425   292     0       4       374
25%-tile:       1       13425   293     0       4       3740
Median:         1       13425   293     0       4       7479
75%-tile:       1       13425   293     0       5       11218
97.5%-tile:     1       13425   294     1       6       14583
Maximum:        3       13425   351     5       9       14956
Mean:   1.00074 13425   292.977 0.0573014       4.57014
# of Seqs:      14956
```

Now we have a customized reference alignment to align our sequences to. The nice thing about this reference is that instead of being 50,000 columns wide, it is now 13,425 columns wide which will save our hard drive some space and should improve the overall alignment quality. We'll do the alignment with align.seqs:

```
mothur > align.seqs(fasta=stability.trim.contigs.good.unique.fasta, reference=silva.v4.fasta)
```

This should be done in a manner of seconds and we can run summary.seqs again:

```
mothur > summary.seqs(fasta=stability.trim.contigs.good.unique.align, count=stability.trim.contigs.good.count_table)

Using 8 processors.

                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1250    10693   250     0       3       1
2.5%-tile:      1968    11550   252     0       3       3222
25%-tile:       1968    11550   252     0       4       32219
Median:         1968    11550   252     0       4       64437
75%-tile:       1968    11550   253     0       5       96655
97.5%-tile:     1968    11550   253     0       6       125651
Maximum:        1982    13400   270     0       12      128872
Mean:   1967.99 11550   252.462 0       4.36693
# of unique seqs:       16426
total # of seqs:        128872
```

So what does this mean? You'll see that the bulk of the sequences start at position 1968 and end at position 11550. Some sequences start at position 1250 or 1968 and end at 10693 or 13400. These deviants from the mode positions are likely due to an insertion or deletion at the terminal ends of the alignments. Sometimes you'll see sequences that start and end at the same position indicating a very poor alignment, which is generally due to non-specific amplification. To make sure that everything overlaps the same region we'll re-run screen.seqs to get sequences that start at or before position 1968 and end at or after position 11550. We'll also set the maximum homopolymer length to 8 since there's nothing in the database with a stretch of 9 or more of the same base in a row (this really could have been done in the first execution of screen.seqs above). Note that we need the count table so that we can update the table for the sequences we're removing and we're also using the summary file so we don't have to figure out again all the start and stop positions:

```
mothur > screen.seqs(fasta=stability.trim.contigs.good.unique.align, count=stability.trim.contigs.good.count_table, summary=stability.trim.contigs.good.unique.summary, start=

mothur > summary.seqs(fasta=current, count=current)

Using stability.trim.contigs.good.good.count_table as input file for the count parameter.
Using stability.trim.contigs.good.unique.good.align as input file for the fasta parameter.

Using 8 processors.

                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1965    11550   250     0       3       1
2.5%-tile:      1968    11550   252     0       3       3217
25%-tile:       1968    11550   252     0       4       32164
Median:         1968    11550   252     0       4       64328
75%-tile:       1968    11550   253     0       5       96492
97.5%-tile:     1968    11550   253     0       6       125439
Maximum:        1968    13400   270     0       8       128655
Mean:   1968    11550   252.463 0       4.36666
# of unique seqs:       16298
total # of seqs:        128655
```

Now we know our sequences overlap the same alignment coordinates, we want to make sure they only overlap that region. So we'll filter the sequences to remove the overhangs at both ends. Since we've done paired-end sequencing, this shouldn't be much of an issue, but whatever. In addition, there are many columns in the alignment that only contain gap characters (i.e. "-"). These can be pulled out without losing any information. We'll do all this with filter.seqs:

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

At the end of running the command we get the following information:

```
Length of filtered alignment: 376
Number of columns removed: 13049
Length of the original alignment: 13425
Number of sequences used to construct filter: 16298
```

This means that our initial alignment was 13425 columns wide and that we were able to remove 13049 terminal gap characters using trump=. and vertical gap characters using vertical=T. The final alignment length is 376 columns. Because we've perhaps created some redundancy across our sequences by trimming the ends, we can re-run unique.seqs:

```
mothur > unique.seqs(fasta=stability.trim.contigs.good.unique.good.filter.fasta, count=stability.trim.contigs.good.good.count_table)
```

This identified 3 duplicate sequences that we've now merged with previous unique sequences. The next thing we want to do to further de-noise our sequences is to pre-cluster the sequences using the pre.cluster command allowing for up to 2 differences between sequences. This command will split the sequences by group and then sort them by abundance and go from most abundant to least and identify sequences that are within 2 nt of each other. If they are then they get merged. We generally favor allowing 1 difference for every 100 bp of sequence:

```
mothur > pre.cluster(fasta=stability.trim.contigs.good.unique.good.filter.unique.fasta, count=stability.trim.contigs.good.unique.good.filter.count_table, diffs=2)
```

We now have 6087 unique sequences. At this point we have removed as much sequencing error as we can and it is time to turn our attention to removing chimeras. We'll do this using the VSEARCH algorithm that is called within mothur using the chimera.vsearch command. Again, this command will split the data by sample and check for chimeras. Our preferred way of doing this is to use the abundant sequences as our reference. In addition, if a sequence is flagged as chimeric in one sample, the default (dereplicate=F) is to remove it from all samples. Our experience suggests that this is a bit aggressive since we've seen rare sequences get flagged as chimeric when they're the most abundant sequence in another sample. This is how we do it:

```
mothur > chimera.vsearch(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.
```

Running chimera.vsearch with the count file will remove the chimeric sequences from the count file. But you still need to remove those sequences from the fasta file. We do this using remove.seqs:

```
mothur > remove.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.fasta, accnos=stability.trim.contigs.good.unique.good.filter.unique.precluster.den
```

Running summary.seqs we see what we're left with:

```
mothur > summary.seqs(fasta=current, count=current)

                Start   End     NBases  Ambigs  Polymer NumSeqs
Minimum:        1       376     249     0       3       1
2.5%-tile:      1       376     252     0       3       2955
25%-tile:       1       376     252     0       4       29543
Median:         1       376     252     0       4       59086
75%-tile:       1       376     253     0       5       88629
97.5%-tile:     1       376     253     0       6       115217
```

```
Maximum:        1       376     256     0       8       118171
Mean:   1       376     252     0       4
# of unique seqs:       2489
total # of seqs:        118171
```

Note that we went from 128,655 to 118,171 sequences for a reduction of 8.2%; this is a reasonable number of sequences to be flagged as chimeric. As a final quality control step, we need to see if there are any "undesirables" in our dataset. Sometimes when we pick a primer set they will amplify other stuff that gets to this point in the pipeline such as 18S rRNA gene fragments or 16S rRNA from Archaea, chloroplasts, and mitochondria. There's also just the random stuff that we want to get rid of. Now you may say, "But wait I want that stuff". Fine. But, the primers we use, are only supposed to amplify members of the Bacteria and if they're hitting Eukaryota or Archaea, then its a mistake. Also, realize that chloroplasts and mitochondria have no functional role in a microbial community. But I digress. Let's go ahead and classify those sequences using the Bayesian classifier with the classify.seqs command:

```
mothur > classify.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.preclust
```

Now that everything is classified we want to remove our undesirables. We do this with the remove.lineage command:

```
mothur > remove.lineage(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.preclus
```

Note is that "unknown" only pops up as a classification if the classifier cannot classify your sequence to one of the domains. Also, keep in mind that if you aren't classifying your sequences using the RDP reference taxonomy, you'll need to customize what the lineages are called. For example, our modified version of the RDP calls mitochondria, "Mitochondria". If you use greengenes, it calls them "f__mitochondria". If you are using greengenes (or SILVA or anything else), you'll need to change these names as appropriate.

If you run summary.seqs you'll see that we now have 2469 unique sequences and a total of 118009 total sequences. This means about 162 of our sequences were in these various groups. Now, to create an updated taxonomy summary file that reflects these removals we use the summary.tax command:

```
mothur > summary.tax(taxonomy=current, count=current)
```

This creates a pick.tax.summary file with the undesirables removed. At this point we have curated our data as far as possible and we're ready to see what our error rate is.

Assessing error rates

Measuring the error rate of your sequences is something you can only do if you have co-sequenced a mock community. This is something we include for every 95 samples we sequence. You should too because it will help you gauge your error rates and allow you to see how well your curation is going and whether something is wrong with your sequencing set up. First we want to pull the sequences out that were from our "Mock" sample using the get.groups command:

```
mothur > get.groups(count=stability.trim.contigs.good.unique.good.filter.unique.precluster.denovo.vsearch.pick.pick.count_table, fasta=stability.trim.contigs.good.unique.good

Selected 64 sequences from your fasta file.
Selected 4048 sequences from your count file.
```

This tells us that we had 64 unique sequences and a total of 4048 total sequences in our Mock sample. We can now use the seq.error command to measure the error rates:

```
mothur > seq.error(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.pr
```

```
Overall error rate:     6.5108e-05
Errors  Sequences
0       3998
1       3
2       0
3       2
4       1
5       0
6       0
7       0
8       0
9       0
10      0
11      2
12      0
13      0
14      0
15      0
16      0
17      0
18      0
19      0
20      0
21      0
22      0
23      0
24      0
25      0
```

```
26      0
27      0
28      0
29      0
30      0
31      1
```

That rocks, eh? Our error rate is 0.0065%. We can now cluster the sequences into OTUs to see how many spurious OTUs we have:

```
mothur > dist.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, cutoff=0.03)
mothur > cluster(column=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist, count=stability.trim.contigs.good.unique.good.filter.unique.prec
mothur > make.shared(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.list, count=stability.trim.contigs.good.unique.good.filter.
mothur > rarefaction.single(shared=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.shared)
```

This string of commands will produce a file for you called stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.groups.rarefaction. Open it. You'll see that for 4048 sequences, we'd have 35 OTUs from the Mock community. This number of course includes some stealthy chimeras that escaped our detection methods. If we used 3000 sequences, we would have about 31 OTUs. In a perfect world with no chimeras and no sequencing errors, we'd have 20 OTUs. This is not a perfect world. But this is pretty darn good!

Preparing for analysis

We're almost to the point where you can have some fun with your data (I'm already having fun, aren't you?). We'd like to do two things- assign sequences to OTUs and phylotypes. First, we want to remove the Mock sample from our dataset using the remove.groups command:

```
mothur > remove.groups(count=stability.trim.contigs.good.unique.good.filter.unique.precluster.denovo.vsearch.pick.pick.count_table, fasta=stability.trim.contigs.good.unique.g
```

OTUs

Now we have a couple of options for clustering sequences into OTUs. For a small dataset like this, we can do the traditional approach using dist.seqs and cluster:

```
mothur > dist.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, cutoff=0.03)
mothur > cluster(column=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist, count=stability.trim.contigs.good.unique.good.filter.unique.prec
```

Clustering stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist

```
iter    time    label   num_otus        cutoff  tp      tn      fp      fn      sensitivity     specificity     ppv     npv     fdr     accuracy        mcc     f1score
0.03
0       0       0.03    2428    0.03    0       2920180 0       26198   0       1       0       0.991108        1       0.991108        0       0
1       0       0.03    516     0.03    22083   2918375 1805    4115    0.842927        0.999382        0.924439        0.998592        0.924439        0.997991        0.8817
2       0       0.03    478     0.03    22788   2918206 1974    3410    0.869837        0.999324        0.920281        0.998833        0.920281        0.998173        0.8937
3       0       0.03    482     0.03    22813   2918219 1961    3385    0.870792        0.999328        0.920844        0.998841        0.920844        0.998186        0.8945
4       0       0.03    482     0.03    22822   2918209 1971    3376    0.871135        0.999325        0.920502        0.998844        0.920502        0.998185        0.8945
```

The alternative is to use our cluster.split command. In this approach, we use the taxonomic information to split the sequences into bins and then cluster within each bin. In our testing, the MCC values when splitting the datasets at the class and genus levels were within 98.0 and 93.0%, respectively, of the MCC values obtained from the entire test dataset. These decreases in MCC value resulted in the formation of as many as 4.7 and 22.5% more OTUs, respectively, than were observed from the entire dataset. The use of the cluster splitting heuristic was probably not worth the loss in clustering quality. However, as datasets become larger, it may be necessary to use the heuristic to clustering the data into OTUs. The advantage of the cluster.split approach is that it should be faster, use less memory, and can be run on multiple processors. In an ideal world we would prefer the traditional route because "Trad is rad", but we also think that kind of humor is funny.... In this command we use taxlevel=4, which corresponds to the level of Order.

```
mothur > cluster.split(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.uniqu
```

```
label   cutoff  numotus tp      tn      fp      fn      sensitivity     specificity     ppv     npv     fdr     accuracy        mcc     f1score
0.03    0.03    524     22532   2918465 2043    3338    0.871   0.9993  0.9169  0.9989  0.9169  0.9982  0.8927  0.8933
```

Next we want to know how many sequences are in each OTU from each group and we can do this using the make.shared command. Here we tell mothur that we're really only interested in the 0.03 cutoff level:

```
mothur > make.shared(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.list, count=stability.trim.contigs.good.unique.good.filter.
```

We probably also want to know the taxonomy for each of our OTUs. We can get the consensus taxonomy for each OTU using the classify.otu command:

```
mothur > classify.otu(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.list, count=stability.trim.contigs.good.unique.good.filter
```

Opening stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.0.03.cons.taxonomy you'll see something that looks like...

```
OTU     Size    Taxonomy
Otu001  12288   Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);"Porphyromonadaceae"_unclassified(100);
Otu002  8892    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);"Porphyromonadaceae"_unclassified(100);
Otu003  7794    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);"Porphyromonadaceae"_unclassified(100);
Otu004  7473    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);Barnesiella(100);
Otu005  7450    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);"Porphyromonadaceae"_unclassified(100);
Otu006  6621    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Porphyromonadaceae"(100);"Porphyromonadaceae"_unclassified(100);
Otu007  6304    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);Bacteroidaceae(100);Bacteroides(100);
Otu008  5337    Bacteria(100);"Bacteroidetes"(100);"Bacteroidia"(100);"Bacteroidales"(100);"Rikenellaceae"(100);Alistipes(100);
...
```

This is telling you that Otu008 was observed 5337 times in your samples and that all of the sequences (100%) were classified as being members of the Alistipes.

Phylotypes

For some analyses you may desire to bin your sequences in to phylotypes according to their taxonomic classification. We can do this using the phylotype command:

```
mothur > phylotype(taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy)
```

The cutoff numbering is a bit different for phylotype compared to cluster/cluster.split. Here you see 1 through 6 listed; these correspond to Genus through Kingdom levels, respectively. So if you want the genus-level shared file we'll do the following:

```
mothur > make.shared(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.tx.list, count=stability.trim.contigs.good.unique.good.filt
```

We also want to know who these OTUs are and can run classify.otu on our phylotypes:

```
mothur > classify.otu(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.tx.list, count=stability.trim.contigs.good.unique.good.fil
```

Phylogenetic

If you are interested in using methods that depend on a phylogenetic tree such as calculating phylogenetic diversity or the unifrac commands, you'll need to generate a tree. This process gets mess as your number of sequences increases. But here's how we'd do it using dist.seqs and clearcut...

```
mothur > dist.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, output=lt, processors=8)
mothur > clearcut(phylip=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.phylip.dist)
```

Analysis

Moving on, let's do something more interesting and actually analyze our data. We'll focus on the OTU-based dataset. The analysis using the phylotype-based analysis is essentially the same. Also, remember that our initial question had to do with the stability and change in community structure in these samples when comparing early and late samples. Keep in mind that the group names have either a F or M (sex of animal) followed by a number (number of animal) followed by a D and a three digit number (number of days post weaning). To keep things simple, let's rename our count, tree, shared and consensus taxonomy files.

```
mothur > rename.file(taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.opti_mcc.0.03.cons.taxonomy, shared=stability.trim.contigs.good.

Current files saved by mothur:
accnos=stability.trim.contigs.good.unique.good.filter.unique.precluster.denovo.vsearch.accnos
column=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist
fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta
group=stability.contigs.good.groups
list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.tx.list
name=stability.trim.contigs.good.names
phylip=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.phylip.dist
qfile=stability.trim.contigs.qual
rabund=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.tx.rabund
sabund=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.tx.sabund
shared=stablility.opti_mcc.shared
taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy
constaxonomy=stability.cons.taxonomy
tree=stability.tre
count=stability.count_table
processors=4
summary=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.summary
```

We now want to do is see how many sequences we have in each sample. We'll do this with the count.groups command:

```
mothur > count.groups(shared=stability.opti_mcc.shared)
```

We see that our smallest sample had 2403 sequences in it. That is a reasonable number. Despite what some say, subsampling and rarefying your data is an important thing to do. We'll generate a subsampled file for our analyses with the sub.sample command:

```
mothur > sub.sample(shared=stability.opti_mcc.shared, size=2403)
```

OTU-based analysis

Alpha diversity

Let's start our analysis by analyzing the alpha diversity of the samples. First we will generate rarefaction curves describing the number of OTUs observed as a function of sampling effort. We'll do this with the rarefaction.single command:

```
mothur > rarefaction.single(shared=stability.opti_mcc.shared, calc=sobs, freq=100)
```

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 2403 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):

```
mothur > summary.single(shared=stability.opti_mcc.shared, calc=nseqs-coverage-sobs-invsimpson, subsample=T)
```

These data will be outputted to a table in a file called stability.an.groups.ave-std.summary. Interestingly, the sample coverages were all above 97%, indicating that we did a pretty good job of sampling the communities. Plotting the richness or diversity of the samples would show that there was little difference between the different animals or between the early and late time points. You could follow this up with a repeated-measures ANOVA and find that there was no significant difference based on sex or early vs. late.

Beta diversity measurements

Now we'd like to compare the membership and structure of the various samples using an OTU-based approach. Let's start by calculating the similarity of the membership and structure found in the various samples. We'll do this with the dist.shared command that will allow us to rarefy our data to a common number of sequences.

```
mothur > dist.shared(shared=stability.opti_mcc.shared, calc=thetayc-jclass, subsample=t)
```

These two distance matrices (i.e. stability.opti_mcc.jclass.0.03.lt.ave.dist and stability.opti_mcc.thetayc.0.03.lt.ave.dist) can then be visualized using the [[pcoa] or nmds plots. 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).

```
mothur > pcoa(phylip=stability.opti_mcc.thetayc.0.03.lt.ave.dist)
```

The output of these commands are three files ending in *dist, *pcoa, and *pcoa.loadings. The stability.an.thetayc.0.03.lt.ave.pcoa.loadings file will tell you what fraction of the total variance in the data are represented by each of the axes. In this case, the first and second axis represent about 45 and 14% of the variation (59% of the total) for the thetaYC distances. The output to the screen indicates that the R-squared between the original distance matrix and the distance between the points in 2D PCoA space was 0.89, but that if you add a third dimension the R-squared value increases to 0.98. All in all, not bad.

Alternatively, non-metric multidimensional scaling (NMDS) tries to preserve the distance between samples using a user-defined number of dimensions. We can run our data through NMDS with 2 dimensions with the following commands

```
mothur > nmds(phylip=stability.opti_mcc.thetayc.0.03.lt.ave.dist)
```

Opening the stability.opti_mcc.thetayc.0.03.lt.ave.nmds.stress file we can inspect the stress and R^2 values, which describe the quality of the ordination. Each line in this file represents a different iteration and the configuration obtained in the iteration with the lowest stress is reported in the stability.opti_mcc.thetayc.0.03.lt.ave.nmds.axes file. In this file we find that the lowest stress value was 0.11 with an R-squared value of 0.95; that stress level is actually pretty good. You can test what hapens with three dimensions by the following:

```
mothur > nmds(phylip=stability.opti_mcc.thetayc.0.03.lt.ave.dist, mindim=3, maxdim=3)
```

The stress value drops to 0.05 and the R2 value goes up to 0.99. Not bad. In general, you would like a stress value below 0.20 and a value below 0.10 is even better. Thus, we can conclude that, NMDS is better than PCoA. We can plot the three dimensions of the NMDS data by plotting the contents of stability.opti_mcc.subsample.pick.thetayc.0.03.lt.nmds.axes. Again, it is clear that the early and late samples cluster separately from each other. Ultimately, ordination is a data visualization tool. We might ask if the spatial separation that we see between the early and late plots in the NMDS plot is statistically significant. To do this we have two statistical tools at our disposal. The first analysis of molecular variance (amova), tests whether the centers of the clouds representing a group are more separated than the variation among samples of the same treatment. This is done using the distance matrices we created earlier and does not actually use ordination.

We can test to determine whether the clustering within the ordinations is statistically significant or not using by using the amova command. To run amova, we will first need to create a design file that indicates which treatment each sample belongs to. There is a file called mouse.time.design in the folder you downloaded that looks vaguely like this:

```
F3D0    Early
F3D1    Early
F3D141  Late
F3D142  Late
F3D143  Late
F3D144  Late
F3D145  Late
F3D146  Late
F3D147  Late
F3D148  Late
F3D149  Late
F3D150  Late
F3D2    Early
F3D3    Early
F3D5    Early
F3D6    Early
F3D7    Early
F3D8    Early
F3D9    Early
```

We can then run amova with this file as follows...

```
mothur > amova(phylip=stability.opti_mcc.thetayc.0.03.lt.ave.dist, design=mouse.time.design)

Early-Late      Among   Within  Total
SS      0.637946        0.59274 1.23069
df      1       17      18
MS      0.637946        0.0348671

Fs:     18.2965
p-value: 0.001*
```

Here we see from the AMOVA that the "cloud" early and late time points has a significantly different centroid for this mouse. Thus, the observed separation in early and late samples is statistically significant. We can also see whether the variation in the early samples is significantly different from the variation in the late samples using the homova command:

```
mothur > homova(phylip=stability.opti_mcc.thetayc.0.03.lt.ave.dist, design=mouse.time.design)

HOMOVA  BValue  P-value SSwithin/(Ni-1)_values
Early-Late      8.54001 <0.001* 0.0659433       0.00724368
```

We see that there is a significant difference in the variation with the early samples having a larger amount of variation (0.066) than the late samples (0.007). This was what we found in the original study - the early samples were less stable than the late samples.

Next, we might ask which OTUs are responsible for shifting the samples along the two axes. We can determine this by measuring the correlation of the relative abundance of each OTU with the two axes in the NMDS dataset. We do this with the corr.axes command:

```
mothur > corr.axes(axes=stability.opti_mcc.thetayc.0.03.lt.ave.pcoa.axes, shared=stability.opti_mcc.0.03.subsample.shared, method=spearman, numaxes=3)
```

This command generates the stability.opti_mcc.0.03subsample.0.03.pick.spearman.corr.axes file. The data for the first five OTUs look like this...

```
OTU     axis1   p-value axis2   p-value axis3   p-value length
Otu001  0.016674        0.943603        -0.357174       0.129681        -0.824046       0.000014        0.898278
Otu002  0.231311        0.326410        -0.625332       0.004194        -0.591031       0.007702        0.890990
Otu003  0.045634        0.846482        -0.495832       0.030853        -0.133392       0.571439        0.515485
Otu004  -0.798596       0.000041        0.118473        0.615218        0.210619        0.371547        0.834357
Otu005  -0.894645       0.000000        0.274803        0.243660        -0.137840       0.558678        0.945995
...
```

This helps to illustrate the power of OTUs over phylotypes since each of these OTUs is behaving differently. These data can be plotted in what's known as a biplot where lines radiating from the origin (axis1=0, axis2=0, axis3=0) to the correlation values with each axis are mapped on top of the PCoA or NMDS plots. Later, using the metastats command, we will see another method for describing which populations are responsible for differences seen between specific treatments. An alternative approach to building a biplot would be to provide data indicating metadata about each sample. For example, we may know the weight, height, blood pressure, etc. of the subjects in these samples. For discussion purposes the file mouse.dpw.metadata is provided and looks something like this:

```
group   dpw
F3D0    0
F3D1    1
F3D141  141
F3D142  142
...
```

We can then run corr.axes again with the metadata option:

```
mothur > corr.axes(axes=stability.opti_mcc.thetayc.0.03.lt.ave.pcoa.axes, metadata=mouse.dpw.metadata, method=spearman, numaxes=3)
```

Opening the file mouse.dpw.spearman.corr.axes, we see:

```
Feature axis1   p-value axis2   p-value axis3   p-value length
dpw     -0.622807       0.004396        -0.087719       0.709773        0.338596        0.150848        0.714304
```

Indicating that as the dpw increases the communities shift to in the positive direction along axis 3.

Another tool we can use is get.communitytype to see whether our data can be partitioned in to separate community types

```
mothur > get.communitytype(shared=stability.opti_mcc.0.03.subsample.shared)

K       NLE             logDet  BIC             AIC             Laplace
1       10903.52        590.94  11470.33        11288.52        10845.20
2       10927.36        435.57  12062.44        11698.36        10436.65
3       11814.80        181.01  13518.16        12971.80        10842.09
4       12621.90        -157.33 14893.53        14164.90        11125.31
5       13540.88        -567.95 16380.79        15469.88        11484.27
```

We see that the minimum Laplace value is for a K value of 2 (10436.65). This indicates that our samples belonged to two community types. Opening stability.opti_mcc.0.03.subsample.0.03.dmm.mix.design we see that all of the late samples and the Day 0 sample belonged to Partition_1 and the other early samples belonged to Partition_2. We can look at the stability.opti_mcc.0.03.subsample.0.03.dmm.mix.summary file to see which OTUs were most responsible for separating the communities:

```
OTU     P0.mean P1.mean P1.lci  P1.uci  P2.mean P2.lci  P2.uci  Difference      CumFraction
Otu005  3.33    0.63    0.41    0.99    10.72   9.42    12.21   10.09   0.14
Otu004  6.17    3.99    3.23    4.93    8.70    7.61    9.95    4.71    0.21
Otu006  5.69    3.99    3.24    4.92    7.48    6.52    8.59    3.49    0.26
Otu010  2.00    0.88    0.60    1.29    3.81    3.23    4.50    2.94    0.30
Otu008  3.91    5.19    4.28    6.29    2.95    2.46    3.53    2.25    0.33
...
```

Again we can cross-reference these OTU labels with the consensus classifications in the stability.opti_mcc.cons.taxonomy file to get the names of these organisms.

Population-level analysis

In addition to the use of corr.axes and get.communitytype we have several tools to differentiate between different groupings of samples. The first we'll demonstrate is metastats, which is a non-parametric T-tetst that determines whether there are any OTUs that are differentially represented between the samples from men and women in this study. Run the following in mothur:

```
mothur > metastats(shared=stability.opti_mcc.0.03.subsample.shared, design=mouse.time.design)
```

Looking in the first 5 OTUs from stability.opti_mcc.0.03.subsample.0.03.Late-Early.metastats file we see the following...

```
OTU     mean(group1)    variance(group1)        stderr(group1)  mean(group2)    variance(group2)        stderr(group2)  p-value
Otu001  0.086642        0.000160        0.004002        0.112360        0.002551        0.016837        0.158841
Otu002  0.073283        0.000282        0.005311        0.079068        0.000468        0.007214        0.512488
Otu003  0.067957        0.000092        0.003040        0.070652        0.000235        0.005108        0.674326
Otu004  0.088306        0.000160        0.004000        0.042956        0.000402        0.006680        0.000999
Otu005  0.110029        0.000558        0.007471        0.014843        0.000954        0.010295        0.000999
Otu006  0.075780        0.000100        0.003157        0.040459        0.000137        0.003902        0.000999
...
```

These data tell us that OTUs 4, 5 and 6 were significantly different between the early and late samples.

Another non-parametric tool we can use as an alternative to metastats is lefse:

```
mothur > lefse(shared=stability.opti_mcc.0.03.subsample.shared, design=mouse.time.design)
```

Number of significantly discriminative features: 85 (92) before internal wilcoxon. Number of discriminative features with abs LDA score > 2 : 85.

Looking at the top of the lefse summary file we see:

```
OTU     LogMaxMean      Class   LDA     pValue
Otu001  5.05061 -
Otu002  4.898   -
Otu003  4.84913 -
Otu004  4.94599 Late    4.39611 0.00044411
Otu005  5.04151 Late    4.68534 0.000441539
Otu006  4.87956 Late    4.19352 0.000238563
...
```

OTUs 4, 5, and 6 are significantly different between the two groups and are significantly elevated in the late samples

Phylotype-based analysis

Phylotype-based analysis is the same as OTU-based analysis, but at a different taxonomic scale. We will leave you on your own to replicate the OTU-based analyses described above with the phylotype data.

Phylogeny-based analysis

OTU and phylotype-based analyses are taxonomic approaches that depend on a binning procedure. In contrast, phylogeny-based approaches attempt similar types of analyses using a phylogenetic tree as input instead of a shared file. Because of this difference these methods compare the genetic diversity of different communities.

Alpha diversity

When using phylogenetic methods, alpha diversity is calculated as the total of the unique branch length in the tree. This is done using the phylo.diversity command. Because of differences in sampling depth we will rarefy the output:

```
mothur > phylo.diversity(tree=stability.tre, count=stability.count_table, rarefy=T)
```

This will generate a file ending in rarefaction.

Beta diversity

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:

```
mothur > unifrac.unweighted(tree=stability.tre, count=stability.count_table, distance=lt, processors=2, random=F, subsample=t)
mothur > unifrac.weighted(tree=stability.tre, count=stability.count_table, distance=lt, processors=2, random=F, subsample=t)
```

These commands will distance matrices (stability.1.weighted.ave.dist) that can be analyzed using all of the beta diversity approaches described above for the OTU-based analyses.

Putting it all together

It is perfectly acceptable to enter the commands for your analysis from within mothur. We call this the interactive mode. If you are doing a lot these types of analysis or you want to use this SOP on your own data without thinking too much there are a couple of other options available.

Batch mode

In the folder that you downloaded from the wiki is a file called stability.batch. If you look at it you'll see all of the commands you ran, but instead of listing out the file names it uses the current option throughout. You can copy and paste from this file and get the same output as we got above. The beauty of the batch mode is that you can run mothur from your command line without much typing. For example you would run the following:

```
$ ./mothur stability.batch
```

Don't enter the "$" that represents the prompt. Sit back and wait and let it rip. This is what we call the batch mode. When we do this it takes about 2.25 minutes to run. The other wonderful thing about this approach is that you can use this very file changing the name of the file you list in make.contigs. You'll also notice that you can enter comments into your batch files using the "#" character.

Command line mode

The third way we have of running mothur is by entering mothur commands directly using the command line mode. This is done like so:

```
$ ./mothur "#make.contigs(file=stability.files, processors=8)"
```

This command will fire mothur up, run make.contigs, and then quit. This is useful for people that want to script commands and go back and forth between different programs. The key ingredients here are the quotes around the commands and the "#" character that tells mothur this is not a batch file. If you really went nuts you could combine commands using ";" characters like so:

```
$ ./mothur "#make.contigs(file=stability.files, processors=8); screen.seqs(fasta=current, maxambig=0, maxlength=275); unique.seqs(); count.seqs(name=current, group=current);
```

Finally, another great resource when running mothur is the logfile. If you go to your folder where you are running mothur, you should find one or more file that looks like mothur.1364488920.logfile. Open that up and you'll see all of the commands you entered and the output that was put to the screen. If anything ever goes wrong and you need to email us, please include this file!

Revisions

- 6/24/19 - Updated to reflect version 1.42.3 outputs.

- This page was last modified on 24 June 2019, at 19:00.