This is a quick guide on how to use Usearch6 to go from fastq files all the way to a table of OTUs. More information can be found on the Usearch project page, http://www.drive5.com/usearch/manual/
Also refer to this page for download and installation instructions. I'm going to assume in this manual that you can call Usearch simply by typing ./usearch6, but this depends on the folder where you have installed it, how you have called it and the folder you're in.

This guide is based on Illumina MiSeq reads and Usearch 6.0.x and Usearch7. Please pay attention to use the right version at each step. It might be possible to replace Usearch6.0.x with Usearch 6.1.x and it might work with 5.x.x or older. All of the steps done in Usearch7 in this manual can be done with either Fastx or Usearch6. Don't try Fastx unless you already have it in your server.

This guide is to be used if your forward and reverse reads do not overlap or if you're not sure if that's the case. If you know you have an overlap, please refer to the Usearch_overlap.pdf manual. If you're still planning your experiment, do your best to have an overlap.

PART I: FILTERING AND MERGING

STEP 1: Quality statistics

In the next step, you will have to decide how many bases you want to cut off from the end of your reads, to simultaneously preserve as many bases as possible and guarantee the quality of your reads. Therefore, in this step, you simply get some statistics on quality. For this, you can use Usearch7 or Fastx. If using Usearch7, the statistcs are explained at length in http://www.drive5.com/usearch/manual/fastq_stats.html

The command:
./usearch7 -fastq_stats <infile> -log <outfile>

Example:
./usearch7 -fastq_stats reads_R1.fq -log reads_R1.stats

STEP 2. Quality trimming

In this step, you cut off bases with low quality and trim all reads to the same length. You can use different lengths for the forward and reverse reads, but you should use the same cut-off for every sample you intend to compare. You can also choose to throw away sequences that have too low quality.

The command:
./usearch7 -fastq_filter <infile> -fastq_trunclen <cutoff> -fastq_maxee <max error number> -fastqout <outfile>

Example:
./usearch -fasq_filter reads_R1.fq -fastq_trunclen 200 -fastq_maxee 3 -fastqout reads_R1.trim.fq
./usearch -fasq_filter reads_R2.fq -fastq_truncqual 180 -fastq_maxee 3 -fastqout reads_R2.trim.fq

STEP 3: Converting to fasta

After the trimming is done, we no longer need the quality information.

The comand:

grep '^@HWI' <infile> -A1 –no-group-separator | sed 's/@/>/' > <outfile>

Example

grep '^@HWI' reads_R1.fq -A1 –no-group-separator | sed 's/@/>/' > reads_R1.fa

STEP 4: Synchronizing

If you've chosen to use the -fastq_maxee option in the previous step, you now have some reads for which you only have the forward reads, others with just the reverse. Therefore, I've written a script that will throw away every read that isn't present in both the forward (R1) and the reverse (R2) files. You can download this script and all other Perl scripts described here from www.biologic.se. This script will automatically choose the name for the output files by adding '.sync.fq' to the end.

The command:

perl resync --fwd=<forward_reads> –rev=<reverse_reads>

Example:

perl resync –fwd=reads_R1.trim.fq –rev=reads_R2.trim.fq

STEP 5: Concatenating reads

In this step we will concatenate the forward and reverse reads into a single artificial amplicon. We will include a separator between them, so they can be split later. Due to downstream applications, this separator can only contain the letters A, C, T, G, N. Keep in mind that the spacer you choose should be rare (long) enough that it's not likely to appear at random in your reads.

The command:

perl cat_reads –spacer=<spacer_string> --file1=<forwad_reads> --file2=<reverse_reads>

Example

perl cat_reads –spacer='NNNNNNN' –file1=reads_R1.fa –file2=reads_R2.fa > reads.cat.fa

PART II: CLUSTERING

STEP 6: File concatenation

In most applications, you want to compare communities from different environments, conditions etc. For this, you have to have the same OTU defined for all samples. Therefore, at this point we concatenate all files.

The command

cat <all_fasta_files> > <outfile>

Example

cat reads1.fa reads2.fa reads3.fa > all.fa

STEP 7: Dereplication

Here we combine all reads that are identical into a single one, keeping track of how many copies there are of each one. This saves a lot of time down the road.

The command:

./usearch7 -derep_fulllength <infile> -output <fasta_file> -uc <uc_file> -sizeout

Example

./usearch7-derep_fulllength all.fa -output all.derep.fa -uc all.derep.uc -sizeout

STEP 8: Sorting

Usearch reads the fasta file in order, assigning each sequence to a cluster as it finds it. Since a sequence that has been read in many copies is most likely to be right, starting by the most frequent sequences and working your way down increases the chances the OTU found are real. We will therefore sort sequences from most frequent to least. At this point you can also throw away sequences that haven't been found a minimal amount of times. Discarding singletons will increase your precision and speed up the process, with a small lost of sensitivity. If you choose not to throw away any sequences, don't write the parameter -minsize.

The command:

./usearch7 -sortbysize <infile> -output <outfile> -minsize <minimal cluster size>

Example:

./usearch7 -sortbysize all.derep.fa -output all.sort.fa -minsize 2

STEP 9: Clustering

Here we cluster our reads by similarity. Usearch uses average-linkage clustering, which means that it is possible that two sequences that are closer to each other than the similarity threshold can still end up in different OTU. One way to minimize this risk is to cluster at a higher similarity first, and then gradually expand these clusters.

You can speed up your process by informing Usearch how many bases it can ignore in the beginning of the read; you can do that for the portion of your forward primer that has no degeneracies.

If you're having memory problems, you can use -cluster_smallmem instead of cluster_fast. This is slightly less accurate.

The command:

./usearch6 -cluster_fast <infile> -id <identity> -uc <uc_file> -idprefix <integer> --centroids <fasta output>

Example:

./usearch6 -cluster_fast all.sort.fa -id 0.99 -uc all.99.uc -idprefix 5 –centroids all.99.fa -sizein

./usearch6 -cluster_fast all.99.fa -id 0.98 -uc all.98.uc -idprefix 5 –centroids all.98.fa -sizein -sizeout

./usearch6 -cluster_fast all.98.fa -id 0.97 -uc all.97.uc -idprefix 5 –centroids all.97.fa -sizein -sizeout

STEP 10: Assigning reads to OTU

We will now look at each of the uc files generated and combine them to determine the number of reads per OTU. At this point, take the opportunity to make a directory just for your new cluster files. This is important downstream.

The command:
perl uc_tables --fine=<finer_clustering> --coarse=<coarser_clustering> > <outfile>

Example:
perl uc_tables –fine=all.derep.uc –coarse=all.99.uc > all.100+99.uc
perl uc_tables –fine=all.100+99.uc –coarse=all.98.uc > all.100+98.uc
perl uc_tables –fine=all.100+98.uc –coarse=all.97.uc > all.100+97.uc

STEP 11: Renaming OTU

Our OTU so far have the name of the read ID of their centroid, which is simply not pleasant. Therefore, we can change their names now to OTU_1, OTU_2 etc. This script can be downloaded from http://drive5.com/python/. You can choose any name for your OTUs, but please use OTU_ if you want to keep following this tutorial.

The command:
python fasta_number.py <infile> <prefix> > <outfile>

Example:
python fasta_number.py otus97.fa OTU_ > otus97num.fa

STEP 12: Splitting the concatenated reads

Now that we've assigned the reads to OTU, we have to split them again to be able to assign them a taxonomy.

The command:
perl uncat_reads –spacer=<spacer_string> --in=<infile> --out1=<fwd_file>
--out2=<rev_file>

Example:
perl uncat_reads –spacer='NNNNNNN' –in=all.97.fa –out1=all_R1.97.fa
–out2=all_R2.97.fa

STEP 13: Classifying OTU

There are many tools for assigning taxonomy to a read. Here we use the SINA classifier. Its online version only accepts 1000 sequences at a time. You can choose to divide your file into chunks of 1000 sequences, and then concatenate the results, or you can download and run the SINA classifier locally: http://www.arb-silva.de/aligner/

STEP 14: Parsing taxonomy

The taxonomy assigned to a forward read won't always agree with the reverse read. What we do here is to take the part in which both agree.

The command:

sina2otu --pair --size –sina=<sina_csv_table> --sina2=<sina_csv_table> > <outfile>

Example:

sina2otu --pair --size –sina=all_R1.97.csv –sina2=all_R2.97.csv > all.97.csv

STEP 15: Creating an OTU table

Here we produce a table with OTUS on the lines, samples on the columns and the classification for each read and the sequence of the representative at the end of each line. You can choose to stop the taxonomy at a certain level – default is 5, or approximately class. If you want the full taxonomy, set the –depth parameter to a very large number.

With online SINA you can choose different databases to use (EMBL, Greengenes, LTP, RDP and Silva, in this order). This script will only consider the last classification for each line, so consider that when choosing which databases to use.

Every classification file that you want included in your OTU table should be in the same folder, and no other files should be in it.

The command:

perl otu_tables --depth=<INTEGER> --samples=<FOLDER> --classification=<SINA_FILE>
--sequences=<FASTA>

Example:

perl otu_tables –depth=5 –samples=all_reads/
--classification=otus97.csv –sequences=otus97.num.fa