

Oct 10, 2019

Default Fuhrman Lab Pipeline for Exact Amplicon Sequence Variant (eASV) Calling from SSU rRNA Amplicons Derived From 515Y/926R Primers

Forked from a private protocol

DOI

dx.doi.org/10.17504/protocols.io.vi9e4h6

Jesse McNichol¹

¹University of Southern California



Jed Fuhrman

University of Southern California

OPEN  ACCESS



DOI: dx.doi.org/10.17504/protocols.io.vi9e4h6

External link: <https://onlinelibrary.wiley.com/doi/full/10.1111/1462-2920.13023>

Protocol Citation: Jesse McNichol 2019. Default Fuhrman Lab Pipeline for Exact Amplicon Sequence Variant (eASV) Calling from SSU rRNA Amplicons Derived From 515Y/926R Primers. **protocols.io** <https://dx.doi.org/10.17504/protocols.io.vi9e4h6>

License: This is an open access protocol distributed under the terms of the **Creative Commons Attribution License**, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited

Protocol status: Working

We use this protocol and it's working

Created: November 12, 2018

Last Modified: October 10, 2019

Protocol Integer ID: 17729

Keywords: 18S, 16S, Fuhrman lab, 515Y/926R, Every base matters, DADA2, qiime2, bbtools, SILVA, PR2



Abstract

Modern sequence analysis methods allow marker gene sequences to be resolved down to single nucleotide variants variously known as Oligotypes, Exact Sequence Variants (ESVs), or Amplicon Sequence Variants (ASVs). Although the naming conventions and algorithms may differ, they all share a common goal of deriving the true biological sequence from noisy data. In this protocol, a general pipeline for determining eASVs (exact amplicon sequence variants) is presented that is suitable for amplicon datasets containing mixed 16S/18S sequences. We have chosen to use the open-source qiime2 framework for this analysis and have added one step at the beginning to split our mixed 16S/18S amplicon libraries using the bbsplit.sh script from the versatile BBtools suite. In our experience, both deblur and DADA2 gave similar results when benchmarked against known sequences in mock communities. Both were able to resolve amplicon sequences from mock communities down to the exact (known) sequences, although DADA2 did produce spurious eASVs under certain circumstances. These spurious eASVs were, however, confined to situations where we knew that sequence data was generated in a non-standard way. Although deblur did not suffer from this same bias, its major disadvantage for our use case was that it discarded too much data (especially for our 18S sequences where up to 90% of sequences would be discarded vs only ~20% for DADA2). Therefore, we chose to use DADA2 with careful validation of the output as described in the protocol as our default pipeline.

Guidelines

Where to do the analysis: I'm assuming you're doing this analysis on kraken.usc.edu, which will make things easier because all the necessary software is already installed. It is no problem to do it on your own computer, but it will just take some more time to install everything.

If doing this on kraken, the easiest way is to use the already built conda environments in /home/conda/miniconda2. These can be modified by anyone who is part of the "anaconda" group (i.e. does not require administrator access). In order to use this, you probably will have to add a line to your ~/.bashrc as follows:

```
#Open an editor for your ~/.bashrc configuration file, for example:
vi ~/.bashrc
#Now go to the bottom of the file, and add the following line:
export PATH="/home/anaconda/miniconda2/bin:$PATH"
#Save and close this file, then run the following command:
source ~/.bashrc
#You will know you have succeeded if the following command:
which conda
#Gives the following output:
/home/anaconda/miniconda2/bin/conda
```

Input: Make sure you have demultiplexed sequences that have not otherwise been processed, and with paired reads in separate files. Pay careful attention to naming the files to something that can be easily parsed later on with shell scripts.

Good example of pairs (a consistent basename):

Sample1_R1.fastq.gz
Sample1_R2.fastq.gz

Bad example (inconsistent basename):

Sample_1.forward.fastq.gz
Sample1.R2.fastq.gz

Good Practice: For posterity, I suggest making a README file in the folder where you're doing the analysis so others can redo your analysis easily in the future or at least understand what you've done. Please include:

- Who you are
- Where you got the data from



- How it was processed in the lab and on the computer
- Any caveats or things that people should know when interpreting the results

Warnings:

1. Software is constantly evolving, and so it's important to know which versions you're using in order for your results to be reproducible. In this protocol, this will be mostly taken care of automatically in the scripts by using conda environments that contain a particular version of qiime2 and other tools. This should make the analysis consistent between different people and robust. But you can create your own environments if you so desire - perhaps a new option or bugfix has come up in qiime2 that you need. In this case, we can just update the scripts globally (for all users) or you can just edit that line yourself.
2. qiime2's default naming convention for eASVs is to create a "hash" of the DNA sequence as the name (think shortform DNA sequence, looks like this: 10ba3fdf5833782847db506db5179c01). The key advantage of this is that if the same eASV sequence is identified independently in two different experiments by two different people, the eASV "hash" will be identical. That would mean you could just do a text search of the IDs (as opposed to something like BLAST) if you want to know if the same thing is coming up across samples/years/experiments. But **this is only true if the sequences have the same length (i.e. AAA and AAAT will have different hashes even if the first three bases are identical). This is important to remember when using deblur trim length parameter, and for our processing pipeline for Eukaryotic sequences where we manually specify trim lengths!** Make sure this option is consistent between the studies you want to compare!
3. If you need to cluster your aESVs afterwards for any reason, qiime2 can do this with VSEARCH but you lose information on the cluster membership. There is a workaround to get this info which is described in the protocol.
4. DADA2 creates a run-specific error model, so this means **it is best to process your samples run by run with DADA2!** Otherwise, you may get unpredictable results due to variations in instrument performance. If you do not have run information (e.g. you downloaded your data from NCBI), you can use Deblur which processes things sample-by-sample.



Before start

Make sure you are familiar with:

- basic bash terminal syntax (some good learning resources: [pcfb](#), [hbb](#))
- how to login to kraken (i.e. `ssh user@kraken.usc.edu`; you can set up an alias in your `~/.bashrc` file as follows to make this easier `"alias kraken='sshpass -p yourpassword ssh yourname@kraken.usc.edu'"` - then all you need to do is type in "kraken" and it will log you in)
- how to use screen (on bash)
- what a conda environment is
- how to get folders on kraken to show up on your local machine (necessary for qiime2 visualizations - most people can help with Macs, ask Jesse or Shengwei for help with doing this for Ubuntu)

It would be helpful if you also know:

- how to make basic bash scripts so you can modify the scripts to suit your needs (if not, it's OK but just stick strictly to the naming conventions described in the protocol)

Some questions to make sure you understand the overall protocol:

- What is a paired-end read?
- Why do pairs overlap for some sequences but not others? Which group overlaps and which one doesn't?
- Why do we bother to trim primers?
- Why do we set the acceptable error rate to 20%?
- Why do we include mock communities in sequencing runs?
- What are the limitations of our analysis pipeline for sequences that do not overlap?

Removing primers and sorting 16s and 18s into two separate pools

- 1
 1. Make sure you have the correct environment configuration for conda by typing "which conda" in your terminal. It should output "/home/anaconda/miniconda2/bin/conda". If not, please refer to Guidelines and Warnings to get set up properly, or install on your own environment if you so desire.
 2. Make a folder with an informative name (hereafter referred to as the "base directory").
 3. Make a README inside the folder with information for posterity (at the same level of detail you'd put in your lab notebook).
 4. Put your raw, demultiplexed files in a folder called "00-raw" within your base directory.
 5. Format should be: forward = **"00-raw/*.R1.fastq.gz"** reverse = **"00-raw/*.R2.fastq.gz"**, rename/gzip as necessary. Don't just rename to gz, otherwise you will have problems later. Running **"gzip *"** in your directory will do the trick. Use "pigz" (parallel gzip) if you're in a hurry. FYI you can always check if a file is gzipped for real by going **"head yourfile"**. If it's just named .gz but not gzipped, it will appear as plaintext. Otherwise it will look like gibberish.
 6. Make sure you have access to the necessary scripts, clone from github as follows (The pipeline may be improved/modified occasionally, so it might be wise to do this every time you start a new analysis. I would also keep copies of the scripts after running them in your working directory, that way you'll be able to exactly reproduce your analysis later if need be or understand what you did later.):

```
git clone https://github.com/jcmcnch/eASV-pipeline-for-515Y-926R.git
```

The github repository is now structured into a single "master" branch that contains 3 variants of the pipeline, each in its own folder (with subfolders for the different parts of the pipeline).

7. You'll be using the scripts in the "DADA2-pipeline" subfolder. Make scripts executable (you can just do this: "chmod a+x ./*" in the cloned directory)
8. Confirm they're executable (check by doing "ls -la")
9. If desired, add the scripts to your \$PATH variable so they will be accessible everywhere when you login (see example below):



```
#add the following line to your ~/.bashrc (can do it in the
terminal with "vi" or with a GUI editor as you prefer)
export PATH="/path/to/dir:$PATH"
#now "refresh" your ~/.bashrc so the changes take effect (do this
in the terminal)
source ~/.bashrc
```

- 2
 1. Run the script "**00-run-cutadapt.sh**" (found in the "00-trimming-sorting-scripts" subdirectory). The script itself can be anywhere, but it must be called from the directory containing the folder "**00-raw**"
 2. Check logs to make sure the number of reads with primers being identified is very high (close to 100%)
 3. Look at the output files, make sure the sizes are reasonable
 4. If script fails, make sure the naming format is correct as specified above and the script is executable
- 3
 1. Run the script "**01-sort-16S-18S-bbsplit.sh**" from your base directory. This will split your mixed reads into 16s and 18s. It also copies some "in-silico mocks" that Liv generated into the analysis. These will come in handy later when you're QC'ing your pipeline.
 2. Check the logs, make sure a very high proportion of reads are sorted (close to 100%, should be 100% for the mocks).
 3. Proceed to individual procedures for each fraction (PROK and EUK)

STEP CASE

For PROK (16S) sequences 15 steps

Now that you've cut your primers and sorted your sequences, the steps below are the pipeline for processing the PROK sequences sorted in the previous step. There is no need to run this prior to the EUK analysis - e.g. you could run one or both at the same time.

Get set up

- 4
 1. Go into your "**02-PROKs**" directory. All scripts for the PROK part of the pipeline (prefaced with "P") should be run from here from now on (they can be found in the "01-prok-scripts" subfolder). One thing I like to do is create a "scripts" folder in this directory and then copy the P* scripts here. That way you can just run "./scripts/your-step.sh".
 2. Look at the files in **02-PROKs/00-fastq**. Make sure they are a reasonable size. If there are mostly 0 size files, then something must have gone wrong with the sorting. Check to make sure you are sorting the correct things. In the PROK sequences, you will have some 0-sized files but they should be rare (i.e. pure EUK mocks).

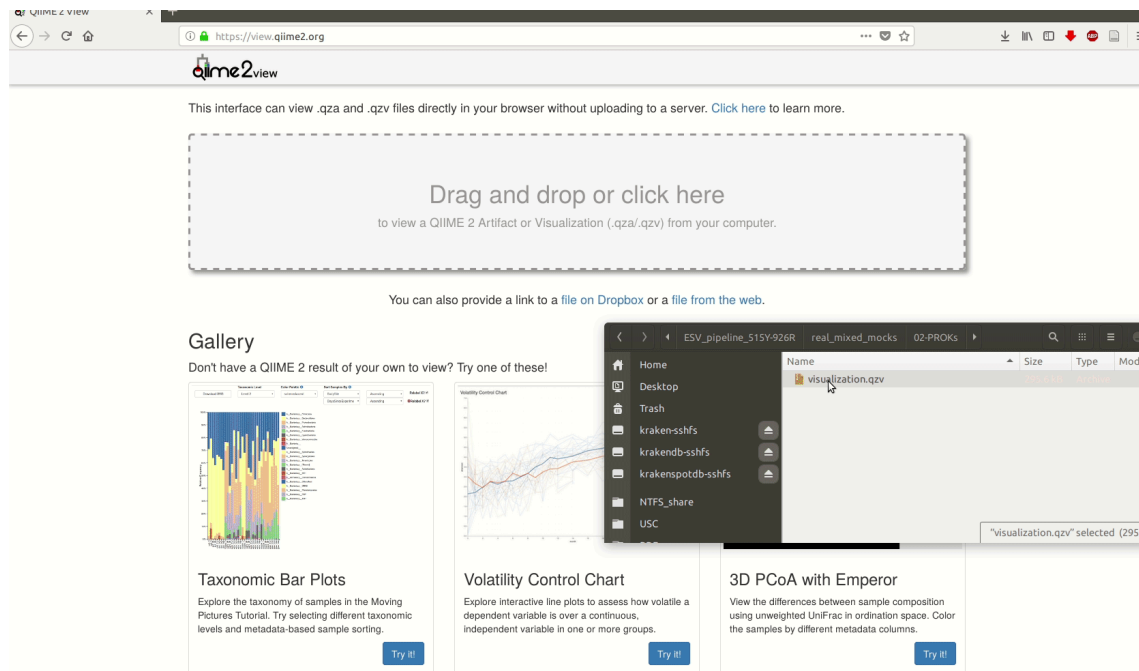
3. Make your manifest. You can do this manually, but I've got a shell script that does the job, which also removes empty files in the process: "**P00-create-manifest.sh**". Just run it from your 02-PROKs folder and double-check the resulting csv file in libreoffice/excel/gnumeric to make sure it looks right. **Any mistakes in your manifest will be carried throughout the rest of the analysis pipeline, so make sure the sample names are all correct! NOTE: Sample ID names should be less than 32 characters.** [Why? The sample-metadata file used to make interactive barplots (Step 11) requires sample ID names less than 32 characters. The Sample ID of the sample-metadata file and the Sample ID of this manifest file must match, so might as well shorten the ID names of the manifest file at this current step. The sample ID names will be used as labels on the barplots].

Import your sequences into a qiime2 artifact

- 5 1. Run the script "**P01-import.sh**" to import your sequences into an "artifact" which is basically an archive file that wraps up all of your raw sequence files and contains information about your analysis (i.e. version of qiime2, commands run etc). On Ubuntu, you can just double-click on it and it opens in archive viewer, but there's not much exciting inside this one except for your raw data.

Visualize sequence quality

- 6 1. Run the script "**P02-visualize-quality_R1-R2.sh**". This creates another artifact, this time of the visualization flavor.
2. Now, visualize by opening up the folder on kraken containing the artifact (if this is not possible, you can always scp the file onto your desktop)
3. You will be using this visualization to decide how much of your R1 and R2 sequences to keep. This is a balancing act between keeping enough of the R2 read (which decreases in quality quite rapidly) to allow merging but not so much that DADA2 will start calling spurious things or discarding your sequences because of with poor-quality data. In my experience, keeping some of the R2 where the quality starts to spread a lot is OK, but **keep the average sequence quality** (i.e. the line in the middle of the box plot) **above 30**. You also need **at least 20 nt overlap** for the merging step. Don't fret too much though, as you can check later with the DADA2 output & mocks to know whether or not your trim length is appropriate.
4. Make a note of these values in your computational notebook for the next step.



Run DADA2!

- 7 1. Run the script "**P03-DADA2.sh**" as follows:

```
P03-DADA2.sh trimleft trimright
```

e.g.

```
$ P03-DAD2.sh 214 187
```

The script will not work unless you put in your trim lengths, and will print them to a file for your future reference. The q2-dada2 plugin is really wrapping a number of separate steps called in the DADA2 pipeline, but is greatly simplified in qiime2 (lucky for us). Liv has tested it inside qiime2 and compared with results from the latest R version of DADA2 and has not noticed any differences. **Remember that DADA2 constructs a run-specific error model, and so we should analyze runs separately!**

Export DADA2 results

- 8 1. Run the script "**P04-export-DADA2-results.sh**". This exports various things from the artifacts into human-readable files. Don't use these for downstream processing, they're just being exported to look at them and make sure everything went well.
2. First, check the "stats.tsv" file found in **03-DADA2d** and make sure most of your sequences make it through the pipeline. If not, you may have over-trimmed and may have to readjust your trim lengths.

Check DADA2 results to look for spurious eASVs or missing sequences

- 9 This step requires you to look carefully at your eASV/OTU table:
1. Open up your **"feature-table.biom.tsv"** file (found in **03-DADA2d**) in a spreadsheet program so you can see it better.
 2. I usually delete the top row, as it's meaningless.
 3. Then find your in-silico staggered mock, and sort the whole spreadsheet in descending order according to that column.
 4. Now, compare your real staggered mock with your in-silico mock. Each row is a distinct eASV, and so you should have the same number of rows (or less) in your real mock as the in-silico mock (which is your "positive control" as they are the exact sequences with perfect quality). Some of the low-abundance eASVs may not be detectable depending on your sequencing depth so that's nothing to worry about. If some of the abundant mocks are missing, this is however a red flag.
 5. Now, you may discover that there are some eASVs in your mocks that don't match up with the in-silico mocks. Don't panic! These can be usually accounted for by sample bleedthrough, but could also be an indication of spurious eASVs being produced by DADA2. We are considering that an eASV is spurious if it has 1-mismatch from the reference sequence across the whole length (we have seen this happen before). In any case, better check. Copy the names of the eASVs into a plaintext file in your **03-DADA2d** folder.
 6. Now grab those sequences. I usually use the command **"seqtk subseq dna-sequences.fasta <your file containing questionable eASVs> > questionable-eASVs.fasta"**. FYI dna-sequences.fasta contains the exact sequences of your eASVs and this command just takes a subset of them based on their fasta identifiers.
 7. Next, blast those eASVs against your mocks. I have a simple wrapper script for this, **"16s-mock-blast.sh"**. Run as follows: **"16s-mock-blast.sh questionable-eASVs.fasta output.tsv"**
 8. Open the output file, and sort by the column "mismatches". Look for ones that have only 1 mismatch. These might be spurious eASVs. If these are not present, that indicates that DADA2 is doing what it's supposed to (at least as far as we can tell from the mocks).
 9. Now, if there are some eASVs present that should not be there but are not a close relative of the mock, then it has to have come from somewhere. There are two possibilities - internal sample bleedthrough or external contamination. The latter seems to be quite rare (although it has apparently happened - saw hydrothermal sequences in SPOT samples once), but the former is fairly common. Bleedthrough can be spotted by scrolling across an eASV row. If the questionable eASV is present in high abundance in other real or mock samples, it's a dead ringer for sample bleedthrough. We don't have a clear recommendation to deal with such sequences at this point, but they seem mercifully rare. Contamination by contrast would be those

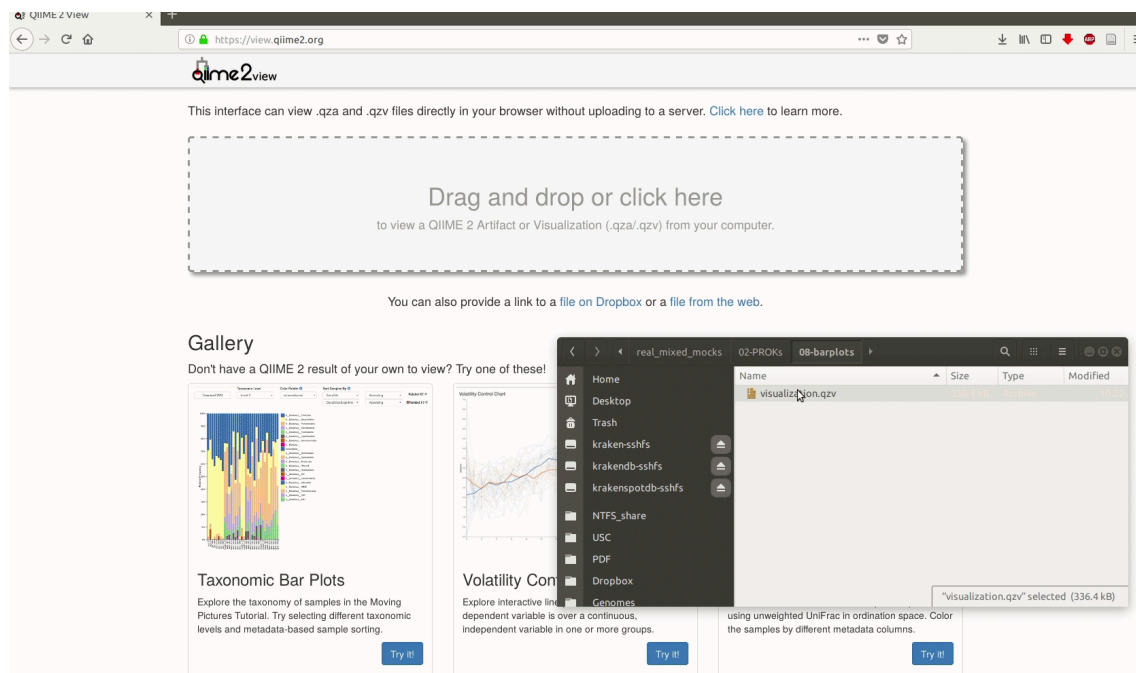
questionable sequences that could not be accounted for by bleedthrough. These may need to be removed.

Classify eASVs against SILVA

- 10 Once you're confident that your mocks look good, and that you don't have any major issues with your denoising, run the script "**P05-classify-eASVs.sh**". This assigns taxonomy to your reads based on a classifier derived from the SILVA 132 database (sliced to the region between our default primers). You can read more about it [here](#). As far as I can tell, it relies on k-mer based classification and so should be appropriate for both prokaryotic and eukaryotic sequences.

Visualize the results with qiime2's built-in interactive barplots

- 11
 1. You first need to make a sample metadata file, which contains any relevant information about your samples that might be useful in plotting. e.g. date, size fraction, etc. Run the following command "**P06-make-sample-metadata-file.py manifest.csv > sample-metadata.tsv**". This will create a template sample metadata file for you. Open it up in your favourite editor and paste in the metadata. Make sure you save it afterwards in the plaintext tab-separated format. Be careful that you've put your metadata in correctly (this is one of those manual steps that is prone to human error).
 2. Now run "**P07-make-barplot.sh**", which will take your abundance information, metadata, and taxonomy and plot it out nicely.
 3. Visualize as before, making sure the results look reasonable. Make sure the top taxa are what you expect. Look for potential contaminants or things that do not belong. Spend some time fussing around with it, sorting by metadata etc:



OPTIONAL: Cluster your aESVs to make OTUs

12 You might want to do some OTU clustering with your eASVs but this is strictly optional. qiime2 has a nice way of handling this functionality (with VSEARCH) this and I suggest doing it after the barplot step if you're going to do it. This way, the subsequent scripts will take the information from the clustering and output them as tables along with the original (unclustered spreadsheet). NB: One of the pieces of information we may wish to have later on is the membership of each cluster. For example, if you want to know how many close relatives a particular eASV has, you would have to know which cluster it is in. By default, this information is not retained in qiime2 but I found a workaround with help from the qiime2 peeps. So it is now possible to do this with the scripts I provide (but it depends on a qiime2 conda environment I've hacked - available on kraken).

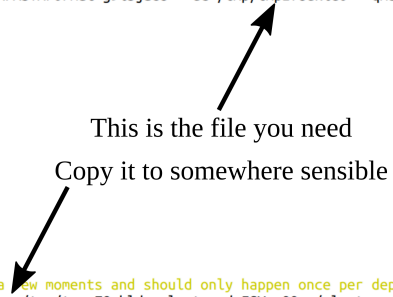
1. Run the script "**P08-optionally-cluster-eASVs.sh <desired cluster level in percent>**". e.g. "**P08-optionally-cluster-eASVs.sh 99**" for 99% clustering. You can run this as many times as you want, it will create a new output folder as long as you supply a different clustering level.
2. Capture the uc file (if desired) as shown in the image below.
3. You can later parse this .uc file with the script "**parse_VSEARCH_cluster_membership_from_UC_file.py**" when you need the membership information. See the help function for this script for more info.

```
(qiime2-2018.8) jesse@kraken:~/ESV_pipeline_515Y-926R/real_mixed_mocks/02-PROKs$ P10a-cluster-ESVs-99pc.sh
QIIME is caching your current deployment for improved performance. This may take a few moments and should only happen once per deployment.
Running external command line application. This may print messages to stdout and/or stderr.
The command being run is below. This command cannot be manually re-run as it will depend on temporary files that no longer exist.

Command: vsearch --cluster_size /tmp/tmpuetbo6ml --id 0.99 --centroids /tmp/q2-DNAFASTAFormat-g9i3jae8 --uc /tmp/tmpz78ukldo --qmask none
--xsize --threads 10

vsearch v2.7.0_linux_x86_64, 1007.9GB RAM, 120 cores
https://github.com/torognes/vsearch

Reading file /tmp/tmpuetbo6ml 100%
9464 nt in 26 seqs, min 364, max 364, avg 364
Sorting by abundance 100%
Counting k-mers 100%
Clustering 100%
Sorting clusters 100%
Writing clusters 100%
Clusters: 26 Size min 1, max 1, avg 1.0
Singletons: 26, 100.0% of seqs, 100.0% of clusters
Saved FeatureTable[Frequency] to: clustered-ESVs-99pc/clustered_table.qza
Saved FeatureData[Sequence] to: clustered-ESVs-99pc/clustered_sequences.qza
QIIME is caching your current deployment for improved performance. This may take a few moments and should only happen once per deployment.
(qiime2-2018.8) jesse@kraken:~/ESV_pipeline_515Y-926R/real_mixed_mocks/02-PROKs$ cp /tmp/tmpz78ukldo clustered-ESVs-99pc/clustered_sequences.uc
(qiime2-2018.8) jesse@kraken:~/ESV_pipeline_515Y-926R/real_mixed_mocks/02-PROKs$
```



This is the file you need
Copy it to somewhere sensible

13 In many cases, you will want to export the data for further analysis or plotting. I like to output the data in a plaintext format (i.e. tab-separated spreadsheet), which can be easily visualized in a spreadsheet program or imported into other software like R/matlab etc. It's also helpful to split the tables into various functional categories (e.g. algae only, heterotrophic prokaryotes only without mitochondria/chloroplasts). These splitting and exporting steps are a little convoluted with qiime2, so I've written some scripts to automate this.

1. First, run "**P09-split-mito-chloro-PhytoRef-reclassify.sh**" which will split out your Chloroplast sequences (based off of initial classifications by SILVA132), reclassify them with PhytoRef, and then generate a merged taxonomy that has SILVA132 classifications for all 16S sequences except for PhytoRef classifications for chloroplasts. **NB-The PhytoRef default Chloroplast taxonomy starts with "Kingdom:Eukaryota" so be aware of that when you look at your exported tables/plots.**
2. Next, run "**P10-generate-tsv-biom-tables-with-taxonomy.sh**", which will export your biom table from the artifact, add the taxonomy information, and then export the whole thing as a .tsv file. It will do this export for every folder in which it finds a feature-table.qza, so it works in batch if you have clustered your eASVs to make OTUs.
3. Check the output in **10-exports/** to make sure everything looks reasonable. Look at the mock community if you have it.
4. Now, the tab-separated file that has been output is in the format of counts. Proportions are typically more informative, so you can run the script "**P10-transform-tsv-to-proportions.sh**" which runs a python script (which needs to be in your \$PATH or in the working directory) that converts your table to proportions and optionally filters out low-abundance eASVs (as with OTUs, there is a long tail of not-very-abundant things). You can get more information about how it works by running "**transform-ESV-tsv-to-proportions.py -h**". By default, the settings I've provided in



the shell script (that runs this python script) will filter out eASVs that are always less than 0.01 % and also keep only the set of eASVs that allow for 99% of the abundance to be recovered in all samples (all of these produce different spreadsheet outputs).

5. Next, you can run "**P12-make-subsetted-barplots.sh**" which will generate barplots with the updated PhytoRef-containing taxonomy (the first barplot in the P07 step above is only SILVA132), and all of the different subsets that you might want to look at. For example, you could look at the community without any chloroplasts or mitochondria, or just the chloroplasts, or just the mitochondria (among other options that you can see in the "10-exports" folder).
6. Finally, if there are certain samples that you would like to exclude from your newly subsetted barplots, like mocks or replicates for example, run "**P13-exclude-samples-from-barplots.sh**". You must first manually generate a sample-to-keep.tsv file which lists the sample IDs that you want to include in all of your barplots. Once generated, the P13 script will continue.

STEP CASE

For EUK sequences 5 steps

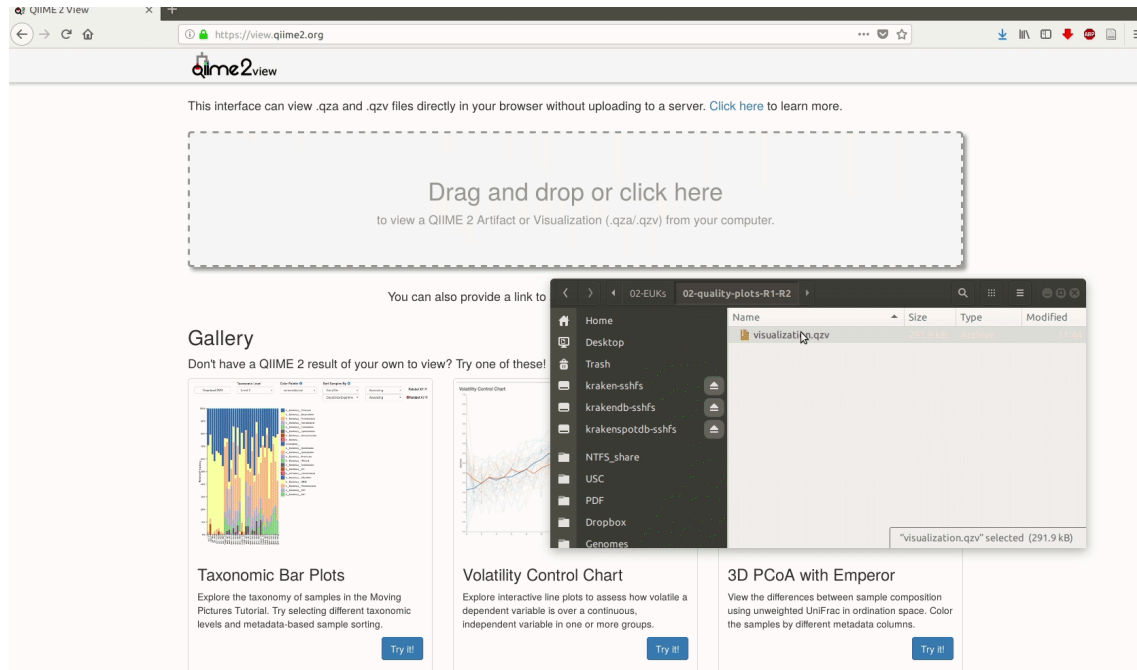
Now we analyze the EUK sequences separately. Since these sequences do not overlap, we have to use a different pre-processing workflow which is nearly identical to what was previously described. Then these pre-processed sequences are put into qiime2 in a similar manner to the PROK sequences.

WARNING: As with the PROK pipeline, you must verify that the real mock sequences come out in the same row as the in-silico mocks and that they are all present. If you trim inappropriately (i.e. keep too much bad sequence in the R2) or mix runs, you may encounter missing mock sequences or spurious eASVs (sometimes caused by the sequences getting truncated at the beginning of the bad quality region). If for whatever reason you don't have real mocks in your run, you will have to be very careful using DADA2 as it is more sensitive to these sort of errors than deblur. You can probably still use it, but proceed with caution as you will have no way to verify your eASVs are called correctly. Deblur can always be a backup option, but it will discard a very large fraction of your sequences (hence why we are currently recommending DADA2, especially for the analysis of EUK sequences).

Visualize the quality of EUK R1 R2 to define trim lengths.

- 14 Since we cannot rely on the read overlapping step to correct errors (as in the PROK analysis), we must make a judgement call on which part of the read is too ugly to keep. We'll use qiime2's beautiful interactive visualization for this (which means we have to import into an artifact first).
 1. As with the PROK analysis, go into your 02-EUKs folder, and run the remaining scripts from here.
 2. First, create your manifest as before by running "**E00-create-manifest-viz.sh**"
 3. Next, import by running "**E01-import.sh**"
 4. Visualize by running "**E02-visualize-quality_R1-R2.sh**".
 5. Now, visualize as before and pick what you think is a reasonable trim length (see gif below for a reminder). This, of course, is somewhat arbitrary. In this gif, I've picked 221 (basically all of the forward read) and 192 (as much of the reverse read as looks

reasonable. However, this may vary from dataset to dataset. In some more recent work, I've seen that a shorter R2 trim length may be necessary (i.e. 120), otherwise your mocks are not recovered. **Whichever trim length you choose, make sure it is consistent within any set of samples that you want to cross-compare and pay close attention to the QC steps where problems may present themselves!**



Cut and paste

- 15 In this step, you are cutting your reads to your chosen trim length and then artificially concatenating them.
1. Run "**E03-bbduk-cut-reads.sh** <fwd trim length> <rev trim length>" to cut your reads.
 2. Run "**E04-fuse-EUKs-withoutNs.sh** " to fuse them together into one read. This differs from what David et al have done previously where an N was added in the middle of the sequence. Remember that when submitting data for publication, something should be added to indicate that these are artificially-fused sequences. It will not cause issues with denoising or taxonomy assignment with the steps outlined in this protocol, but would cause problems if you were using an alignment-based taxonomic assignment algorithm.

Run DADA2

- 16 Because the last few steps were done outside the qiime2 environment (we only imported at the beginning for visualization), we now have to import as before:



1. Run "**E05-create-manifest-concat.sh**" to automatically create your manifest.
2. Run "**E06-import-concat.sh**" to import your sequences.
3. Run "**E07-visualize-quality-single-seqs.sh**" to get a visualization artifact for your merged sequences. Take a look as before, you should see a drop-off in the middle (corresponding to the end of the R2) with higher qualities at both ends. If it doesn't look like this, something is wrong.
4. Run "**E08-DADA2.sh**". There is no need to supply any arguments, as the sequences are all the same length now and so do not require trimming as with the PROKs. For your interest, we're running the "**qiime dada2 denoise-single**" command here.

Export the results

- 17 Now, let's take a look at the results, paying close attention as above to the position of the eASVs compared to the in-silico mocks.
1. Run "**E09-export-DADA2-results.sh**" to export as before.
 2. Take a look at your **stats.tsv** first to make sure most sequences make it through to the end. We've been getting a healthy ~80% so far. This is much better than deblur, which only retains ~10% unless you cut off most of the R2.
 3. Now, check the eASV table as before. You could have two potential issues here if the rows from the real mocks don't match up with the in-silico mocks. One is that you have spurious eASVs as above. The other is that DADA2 might truncate the reads in the middle where the tail of the R2 is. This can result in basically a R1-only eASV. Both can be identified as described above in the PROK protocol except using the "**18s-mock-blast.sh**" script instead of the 16s version of that script (it's the same script, just blasting different databases).

Create barplots, cluster, export, etc

- 18 The remaining steps are essentially the same as for the PROK pipeline, so just follow the numbering in the scripts to guide you through the rest!