
From Fastq sequences to ready to use ASV's and OTU's: how to run the dada2-based pipeline remotely or locally

Contents

Overview and output.....	1
When to overwrite=TRUE or FALSE.....	3
Organising data on the server	4
Running on ALICE	6
Running on a local machine	8

Overview and output

The goal of this pipeline is to process raw single sample sequence data FASTQ files to obtain ASV and OUT tables along with a taxonomy table defining the provenance of the detected sequences. This is achieved with the dada2 R package pipeline, which has been further expanded to deal with variable amplicon lengths of ITS regions, binned quality scores of Novaseq and to create OUT tables of any required similarity percentage.

Naturally, anyone can run dada2 functions locally and follow along with the online tutorials and forums describing OUT construction and learnError adaptations. For larger datasets run times can become very long on personal computers however. Therefore, all functions were collected in a single script that can be run both locally and on a remote server such as ALICE ([link here](#)).









In order to keep flexibility, the pipeline was split into two main steps. The first step does only basic pre-processing and quality benchmarking, while the second step does the rest of the processing. The quality investigation of the first step should support the user in selecting filtering thresholds appropriate for their data. For example, most datasets will be fine with a maximum estimated error of 2, but not all sequence run reads can be truncated at the same base pair length. Some amplicons may be longer or shorter and base pair quality values may drop more towards the end of the read in one run than in the other.

Beyond filtering thresholds, there are also other choices to be made that are influenced by the intended research question. For example, pooling samples during dereplication and merging will increase detection of rare sequences, but it will also increase run times and decrease differences between samples. This makes it ideal for questions concerning rare genera or species. On the other hand, unpooled samples show differences between samples better and takes less time to run, which is good when rare sequences are of little interest or more likely to be background noise. Dada2 also offers an intermediate option with pseudo pooling that shows some but not all rare sequences and better preserves sample differences.

As a safeguard and manner of transparency, the pipeline generates a lot of files and reports. Some of these can be reused when rerunning the same dataset with different settings (i.e. error models for dereplicating and merging pooled or unpooled) and others can be used to continue the pipeline after a crash, timeout or other interruption. These files also allow the user to investigate what the pipeline did at every step and work manually from there if necessary. For an overview of what files are created when, and all processing sub steps in general, view Figure 1.

Step 1

Setup FASTQ files and filter thresholds

1. Remove all reads with one or more N
→  ./Filtered_N_Reads/
 ./Reports/Report_Filtered_Reads.txt
2. Detect given primers in correct orientation
3. Remove detected primers with Cutadapt
→  ./Filtered_Primers/
4. Generate quality plots of both forward and reverse reads
→  ./Reports/Forward_Read_Quality.pdf
 ./Reports/Reverse_Read_Quality.pdf
 ./Reports/FW_Read_Length_Distribution.pdf
 ./Reports/RV_Read_Length_Distribution.pdf
5. Try several filtering thresholds on a subset of 12 samples
→  ./Reports/Filter_Tests.txt

Step 2

Process FASTQ files and create ASV/OTU and taxon tables




















1. Filter all reads with the given thresholds
→  ./Filtered_Reads/
Updates  ./Reports/Report_Filtered_Reads.txt
2. Create updated quality plots
→  ./Reports/Filtered_Forward_Read_Quality.pdf
 ./Reports/Filtered_Reverse_Read_Quality.pdf
3. Error learning
→  ./Reports/Forward_Error_Model.rds
 ./Reports/Reverse_Error_Model.rds
4. Plot error models
→  ./Reports/Forward_Error_Model.pdf
 ./Reports/Reverse_Error_Model.pdf
5. Error correct and dereplicate (here with no sample pooling)
→  ./Reports/Pool_FALSE_Error_Corrected_FW.rds
 ./Reports/Pool_FALSE_Error_Corrected_RV.rds
6. Merge paired end reads into raw sequence table
→  ./Reports/Pool_FALSE_Raw_Sequence_Table.rds
Updates  ./Reports/Report_Filtered_Reads.txt
7. Remove sequence chimera's
→  ./Reports/Pool_FALSE_NonChim_Sequence_Table.rds
 ./Reports/Pool_FALSE_NonChim_Sequence_Table.csv
Updates  ./Reports/Report_Filtered_Reads.txt
8. Collect all read filtering in an overview
→ ./Reports/Pool_FALSE_Final_Filter_Report.xlsx
9. Fungi only: Create OUT table from ASV (here with 97% similarity)
→  ./Reports/Pool_FALSE_Alignment.rds
 ./Reports/Pool_FALSE_0_97_OTU_Table.rds
10. Assign taxonomy (here with /users/Unite_Full.fa.ga)
→  ./Reports/Pool_FALSE_Taxon_Table_Unite_FULL.rds
 ./Reports/Pool_FALSE_Taxon_Table_Unite_FULL.xlsx

Figure 1: Overview of all pipeline steps and when what files are generated. Note how sample pooling and used database influence the file names. This is to allow multiple parameter settings to be run in parallel without having to manually organise any pipeline files.

When to overwrite=TRUE or FALSE

Both step 1 and 2 of the pipeline have the argument `overwrite` as a contingency. This bit is to explain what it does and when to use it.

In essence, `overwrite` tells the pipeline to either generate all files even if some are already present (TRUE) or use what is already present to skip steps (FALSE). The latter is especially useful if the script crashed or timed out halfway and you want to continue where you left off. In addition, step 2 has a lot of options that can change files near the end of the pipeline, such as sample pooling and database name. It makes little sense to refilter all FASTQ files on the same thresholds just because another taxonomy database is used, so the pipeline cuts corners where it can to reduce running times.

A couple of things to keep in mind:

1. In step 1, the *Removed_N_Reads* folder is required to create the *Removed_Primers* one. If the former doesn't exist, the latter cannot be made.

If both are present, `overwrite=FALSE` will create new quality plots and filter benchmarks

2. In step 2, file names are inferred from sample pooling and the database name. Keep in mind that databases in different folders but the same base name will both use the same files. Therefore take care to not use databases of the same name when they are in fact different.

For example. `/home/USER_1/Unite_database.fa` and `/home/USER_2/Unite_database.fasta` both have the same basename (`Unite_database`). This means that both will use the file name if both have the same sample pool setting: `./Reports/Pool_FALSE_Taxon_Table_database.rds`.

3. If you are unsure if previously generated files are complete, it is best to just set `overwrite=TRUE`.
4. If you want to start at a specific step, you can go to the FASTQ folder and manually delete the files that have to be updated. Keep in mind that the pipeline doesn't automatically update all downstream files if an upstream file is missing though.
For example, deleting the raw sequence table will have the pipeline generate a new one, but it won't force it to use that new table to overwrite an existing downstream clean sequence table.

Thus, delete all files that need to be changed, not only the one upstream.

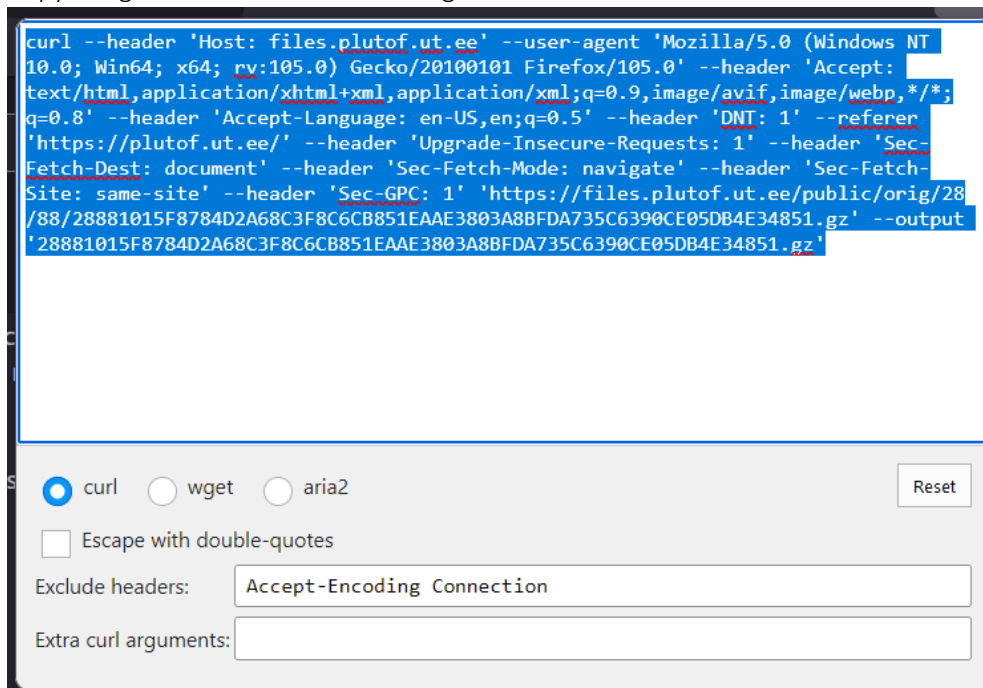
Organising data on the server

Dada2 compatible databases:

<https://benjineb.github.io/dada2/training.html>

If you want to fetch another dada2 compatible database, but don't know how to download that one to the server or can't find the download link. There is a generic method of easily knowing what to enter into the terminal:

1. Install the cliget extension on Chrome or Firefox.
2. Go to the desired database and start a download to your local machine. This can be cancelled as soon as it starts downloading data (not if it is preparing the download)
3. Cliget should have picked up on that download and generated an appropriate command, so go to its extension notification and select your download.
4. Copy the given command of either wget or curl and enter it into the ALICE terminal



5. In the example above, the downloaded file has the weird name '28881015F8784D2A68C3F8C6CB851EAAE3803A8BFDA735C6390CE05DB4E34851.gz', shown after --output.

Once the download is finished you can change that name to whatever, so long as the extension is correct. In this case the file was actually a .fa.gz, but only got the .gz extension, so the downloader is not always right.

Downloading Silva database:

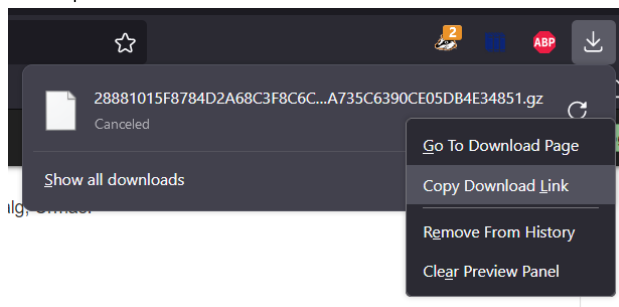
1. Go to the archive of Dada2 formatted Silva references:
<https://zenodo.org/record/4587955>
2. Pick the most recent file for the most up to date reference database.
3. Once you have found the correct file, right click it and select 'copy URL'. Then go to the ALICE terminal and type

```
curl YOUR_URL > desired_file_name.fa.gz
```

This will start the download to you current working directory (should be the home folder by default) and save it under the defined file name.

Downloading Unite databases:

1. Go to the Unite database and find the desired release under *General FASTA releases*:
<https://unite.ut.ee/repository.php>
2. Follow the file DOI link to the page with release details and at the bottom a row 'Downloads' with the corresponding file. Click on this file to start the download to your local machine and immediately cancel it, as we want it on the server, not your personal computer.
Go to the downloads tab on your browser and select 'copy download link'. Below is an example from firefox:



This gave the following link:

<https://files.plutof.ut.ee/public/orig/28/88/28881015F8784D2A68C3F8C6CB851EAAE3803A8BFDA735C6390CE05DB4E34851.gz>

3. Paste the following into the command line and replace the web address and example file name with the desired names. Note that the downloaded file name should always have the `curl YOUR_URL > desired_file_name.fa.gz`
This will start the download to you current working directory (should be the home folder by default) and save it under the defined file name.

Note that Unite sometimes provides tar files, recognisable by the .tgz extension. In that case the extension shown in the example above (.fa.gz) is incorrect and should be changed. Then run this additional line to untar the file and obtain a decompressed fasta:

```
tar -xzf desired_file_name.tgz
```

An example of a .tgz Unite can be found [here](#):

Formats	• application/gzip
Subjects/keywords	UNITE
Licence	Attribution-ShareAlike (CC BY-SA)
Citation	Abarenkov, Kessy; Zirk, Allan; Piirmann, Timo; Pöhönen, Raivo; Ivanov, Filipp; Nilsson, R. Henrik; Kõljalg, Urmas (2021): UNITE general FASTA release for Fungi. UNITE Community. 10.15156/BIO/1280049
Related identifiers	• IsNewVersionOf: 10.15156/BIO/786368 (Dataset)
Alternate identifiers	
Downloads	sh_general_release_10.05.2021.tgz

Running on ALICE

1. Open the SLURM script you wish to run (either 'SLURM_Step_1.slurm' or 'SLURM_Step_2.slurm') with your preferred text editor and change the arguments marked in red to whatever is correct for your data.

Arguments highlighted with blue are optional and should be changed when the default time running time is too short. Keep in mind that the partition name has to change as well at certain time limits, since each has their own maximum run time:

cpu-short = max. 4 hours
 cpu-medium = max. 24 hours (1 day)
 cpu-long = max. 168 hours (7 days)

For step 1:

```
#!/bin/bash
#SBATCH --job-name=l6S_Processing # Job name
#SBATCH --mail-user=<d.remarque@hotmail.com> # Where to send mail
#SBATCH --mail-type=FAIL,END # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --partition=cpu-short # Partition to use for the analysis
#SBATCH --time=01:30:00 # Time limit hrs:min:sec
#SBATCH --ntasks=1 # Number of tasks per node
#SBATCH --cpus-per-task=16 # Number of cpus requested per task
#SBATCH --export=none # Recommended by ALICE Staff for a clean working environment

# INPUT
script_dir=/home/s1838768/Scripts/ # Location of all pipeline R scripts
fastq_dir=/data/s1838768/AllRawITS/ # Location of raw fastq files

Forward_Primer_Sequence=GTGARTCATCRARTYTTTG # Known forward primer sequence
Reverse_Primer_Sequence=CCTSCSCTTANTDATATGC # Known reverse primer sequence
Overwrite_Files=TRUE # Should previously generated files be overwritten (see manual for explanation)
Delete_Files=TRUE # Should N filtered reads be deleted afterwards to save space?

# DO NOT CHANGE
# Load modules and find path
module load R/3.6.2-fosscuda-2019b
module load cutadapt/1.18-GCCcore-8.2.0
cutadapt_path=which cutadapt

# Find script
cd $script_dir

# Execute RScript
time Rscript ./Primer Removal and Qual Plot.R --fastq_folder $fastq_dir --fw_primer $Forward_Primer_Sequence --rv_primer $Reverse
```

For step 2:

```
#!/bin/bash
#SBATCH --job-name=ITS_F_Processing # Job name
#SBATCH --mail-user=<d.remarque@hotmail.com> # Where to send mail
#SBATCH --mail-type=FAIL,END # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --partition=cpu-short # Partition to use for the analysis
#SBATCH --time=04:00:00 # Time limit hrs:min:sec
#SBATCH --ntasks=1 # Number of tasks per node
#SBATCH --cpus-per-task=6 # Number of cpus requested per task
#SBATCH --mem-per-cpu=1500MB
#SBATCH --export=none # Recommended by ALICE Staff for a clean working environment

# INPUT
script_dir=/home/s1838768/Scripts/ # Location of all pipeline R scripts
fastq_dir=/data/s1838768/AllRawITS/ # Location of raw fastq files
database=/home/s1838768/Unite_DB_FULL.fa.gz # Location of reference database (can be .gz)

Is_Novaseq=TRUE # The sequencing method was Novaseq (TRUE) or something else like MiSeq (FALSE)

maxEE=2 # Maximum estimated error in a read before it is discarded
truncLen_forward_reads=220 # Maximum bp forward read length before truncation. Shorter reads are discarded
truncLen_reverse_reads=220 # Maximum bp reverse read length before truncation. Shorter reads are discarded
minLen_read=20 # When truncLen=0 (turned off). How long should the minimal read bp be
truncQ=2 # When truncLen=0 (turned off). Truncate reads when they drop below this quality score

Pool_Samples=FALSE # When identifying ASVs, should samples be pooled? TRUE = high rare sequence sensitivity, FALSE = hi

otu=0.97 # Percentage similarity for OTU's. 1 = no OTU, 0.97 = 97% OTU
otu_abundance=TRUE # Whether the OTU representative sequence is most abundant (TRUE) or consensus (FALSE)

Overwrite_Files=FALSE # Should previously generated files be overwritten (see manual for explanation)

# DO NOT CHANGE
# Load modules
module load R/3.6.2-fosscuda-2019b

# Find script
cd $script_dir

# Execute RScript
time Rscript ./DADA2_Step_2.R --fastq_folder $fastq_dir --database $database --novaseq $Is_Novaseq --maxEE $maxEE --truncLen_fw $
```

2. With the correct arguments, save the file and make sure the encoding is UTF-8. If the encoding is something else like UTF-8 BOM, then SLURM will not recognise it as a batch file and throw the following error:

```
sbatch: error: This does not look like a batch script. The first
sbatch: error: line must start with #! followed by the path to an interpreter.
sbatch: error: For instance: #!/bin/sh
```

3. Now submit the job with the following command (change USER to your username of course):

```
sbatch /home/USER/SLURM_Step_1.slurm
```

4. Optional: Check the jobstatus with the universal command

```
squeue -u $USER
```

This shows the following, with the most important part being ST (state), which is either running (R) or pending (PD). Once the job starts you can see how long it has been running under time.

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
730269	cpu-medium	16S_Proc	s1838768	R	19:48	1	node003
730268	cpu-medium	ITS_F_Pr	s1838768	R	19:54	1	node013

5. Optional: Check the job log during the run:

To keep the pipeline transparent rather than a black box that may or may not be close to finishing, output is given in a special log file. This file always has the same naming structure of slurm_JOBID.out and can be read with any text editor.


Running on a local machine

1. Open the R script DADA2_Step_1.R or DADA2_Step_2.R in R studio.
2. Set the working directory to the location of the R scripts folder on your local computer. Please make sure all 4 scripts (DADA2_Step_1.R, DADA2_Step_1_Source_Code.R, DADA2_Step_2.R, DADA2_Step_2_Source_Code.R) are in that directory.

3. Unhash and change all arguments as required. If the default argument settings are fine then it does not have to be unhashed. As an example for step 1, the fastq_folder, cutadapt path and primer sequences were changed, but the overwrite and delete arguments were left default:

```
#####  
#                               DADA2 Pipeline STEP 1                               #  
#####  
message(  
"  
This is the first step in the DADA2 pipeline to go remove primer/adaptor sequences and  
to explore FASTQ file quality.  
Please ensure the R working directory is set to the R script locations!  
")  
  
#### Collect all the functions ####  
source('DADA2_Step_1_Source_Code.R')  
  
#### Collect the given arguments ####  
arguments <- Argument_Parsing()  
## Unsure what an argument is/does? Unhash and run below for help:  
# Argument_Parsing(show_help=T)  
  
#### When running locally, you can input your directories below and unhash ####  
  
arguments$fastq_folder <- '/C/s1838768/FASTQ_Files/'  
arguments$cutadapt      <- '/C/Program Files/cutadapt/cutadapt.exe'  
arguments$fw_primer      <- 'GTGYCAGCMGCCGCGGTAA'  
arguments$rv_primer      <- 'GGACTACNVGGGTWTCTAAT'  
  
## All other arguments can be left unchanged if the defaults below are okay  
## Otherwise change them as necessary and unhash  
  
# arguments$overwrite    <- TRUE  
# arguments$delete       <- TRUE  
  
#### Run the code ####  
main(opt=arguments)
```

As a tip: If you don't know what the arguments do, run the following in the R console for help:
source('DADA2_Step_1_Source_Code.R')
Argument_Parsing(show_help=T)

4. When everything is filled in correctly, run the entire script by clicking the  Run button on the top right of the Rstudio script window or by pressing Ctrl+A followed by Ctrl+Enter.
5. All pipeline progress can be followed in the Rstudio console window, where the pipeline log is automatically shown in real time.