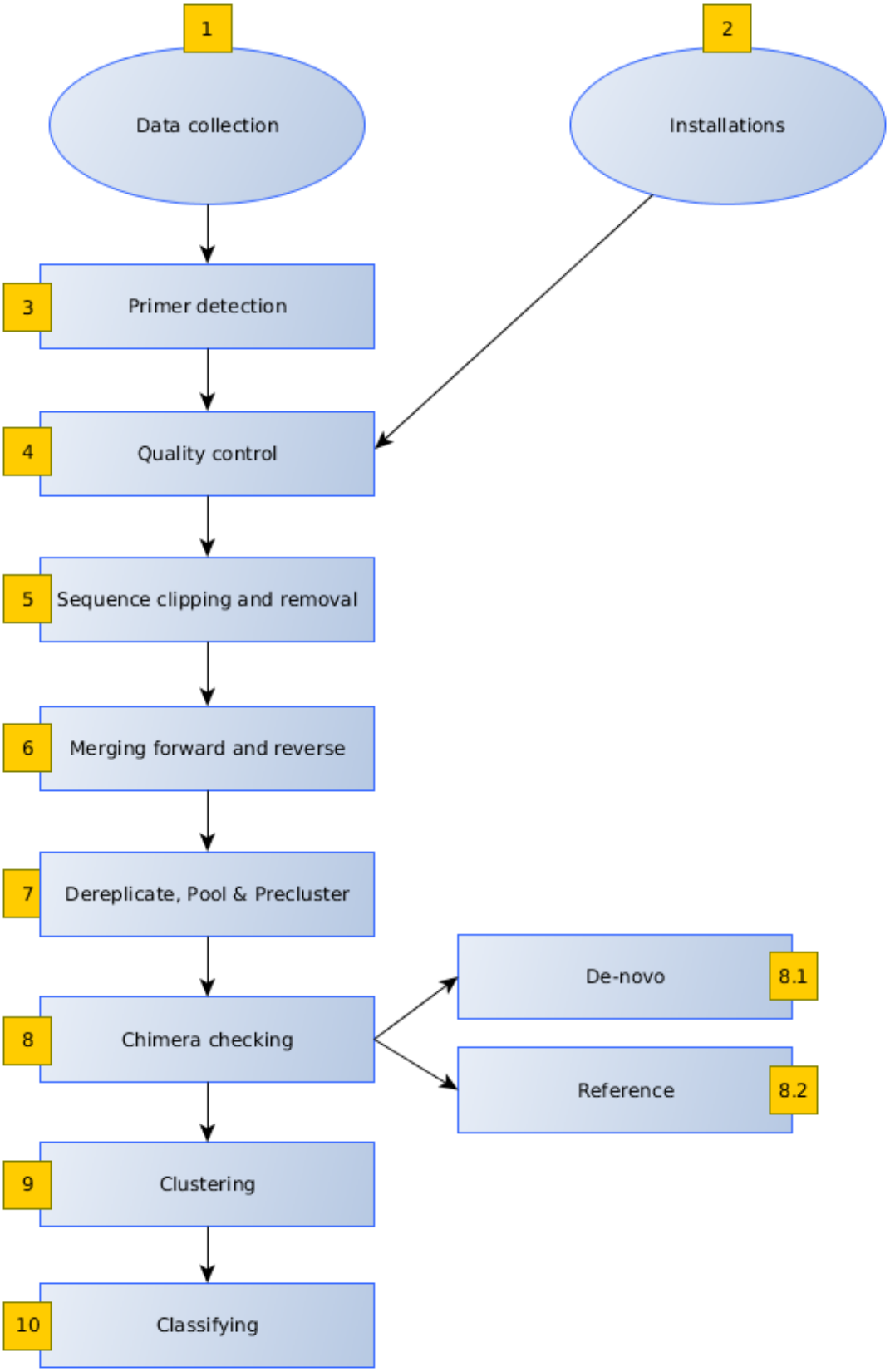
Amplicon sequencing - 16S rRNA

Table of Contents


1
Data collection
2
Installations
3
Primer detection
4
Quality control
5
Sequence clipping and removal
6
Merging forward and reverse
7
Dereplicate, Pool & Precluster
8
Chimera checking
De-novo
8.1
Reference
8.2
9
Clustering
10
Classifying

The Amplicon exercise

This exercise contains over 50 questions.

This is an individual assignment, but conversations and discussions are encouraged.

Record your answers to a **Word file** and submit it to the teacher when you are done. **Note**, None of your answers should be longer than 3-5 lines. Longer lines will lose points.

Provide a collection of your scripts in a single text file with a short annotation before every section.

1 Aim, data, and installation

1.1 Aim

The aim of this exercise is to learn how to deal with amplicon sequencing data, in this case, 16S rRNA gene data. We will start with `fastq` files containing DNA fragments obtained from the hindgut of termites and then proceed to a final visualization of the structure of bacterial communities in different samples. We will compare our results with the ones from the published study. The analysis involves many steps and we will go through them one by one.

1.2 Data

1.2.1 Fastq

We will work with Illumina data from the study of Benjamino et al. (2016) http://www.ncbi.nlm.nih.gov/pubmed/26925043. The study explores the bacterial community in the hindgut of the termite *Reticulitermes flavipes*. The microbes in the hindgut help the termite digest the lignocellulose from the "wood food."

The study's data are found at the European Nucleotide Archive (under *Study*):
http://www.ebi.ac.uk/ena/data/view/ERS901611
These data take time to download, so copy them from Canvas or the server:
/resources/binp28/Data/AmpliconData.tar.gz (uncompress in your folders):

```
cd
mkdir -p Amplicon
cp -r /resources/binp28/Data/AmpliconData.tar.gz ~/Amplicon
```

There are 118 `fastq` files in total corresponding to 59 runs (since we have forward and reverse reads).

Read the paper and answer the following questions:

1. How many hindgut microbiota termite samples were studied (colony + caste)?
2. What is the most abundant phylum in the termite microbiota?

3. How does the hindgut microbiota differ amongst castes?

4. How much space do the files take after decompression (use disk usage `du`)?

The samples were collected from eight termite colonies in the U.S., but we will work with seven of them. To show that European quality can be poor or that Europeans don't have a clue where the U.S.A. is on the map, look at the geolocation according to EBI (under *Additional Attributes*): http://www.ebi.ac.uk/ena/data/view/ERS901665

5. The study was done in the USA, but EBI geographical coordinates point to a different country. Which country is it? What does that tell you about the reliability of the data?

Samples were taken from workers, soldiers, and alates (winged termites). We will only work with worker termites. We are interested to see if we find different bacterial flora in the various colonies. From each location, several individuals have been sequenced, so we have to pool the individuals. To make things even worse, there may be more than one run per individual, and we want to work with only one sample per individual.

6. How would you remove multiple copies of a sample?

Our goal is to replicate the results presented in Figure 2 of the paper.

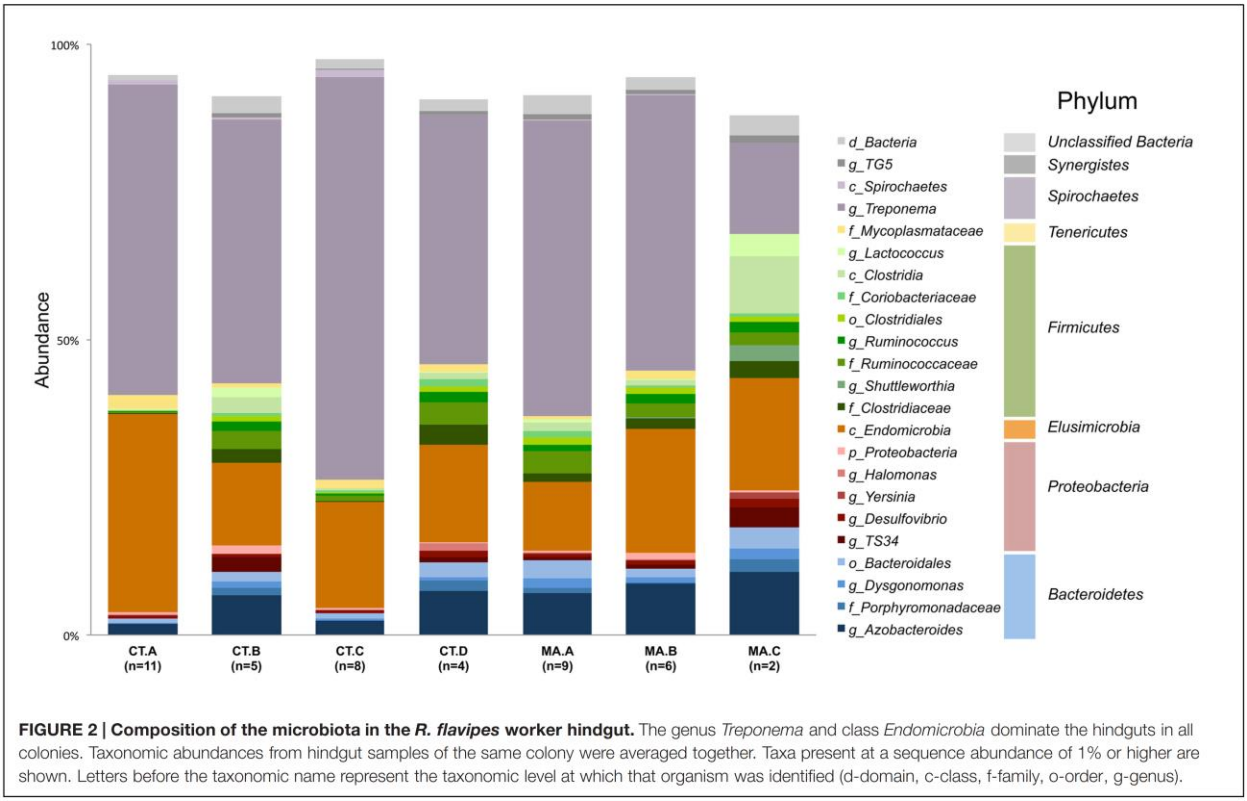


FIGURE 2 | Composition of the microbiota in the *R. flavipes* worker hindgut. The genus *Treponema* and class *Endomicrobia* dominate the hindguts in all colonies. Taxonomic abundances from hindgut samples of the same colony were averaged together. Taxa present at a sequence abundance of 1% or higher are shown. Letters before the taxonomic name represent the taxonomic level at which that organism was identified (d-domain, c-class, f-family, o-order, g-genus).

Each bar is a colony/sample site where `CT` stands for Connecticut and `MA` for Massachusetts. Note that *n* should be `10` in colony *`CT.A`* and `8` in colony *`MA.A`*.

We want to produce **two `fastq` files per colony** (one forward and one reverse). We will do that by concatenating the `fastq` files for all the runs originating from that colony. The mapping between runs and colonies is not easy to extract from the website, so we did that for you by writing an XML parser, which produced two `fastq` files per colony.

> Please review the document XML parsing and data format - 2021.docx.
>
> The document describes how these files are processed for each colony (`CT`).
>
> You do not need to do anything, because we already did this step for you.

The files are available from Canvas (Colonies.tar.gz) or are saved in the directory `/Amplicon/Raw/Colonies`.

`Check that` the filenames to `CT_A_1.fastq` and `CT_A_2.fastq`. Notice that we do not use dots in the filename (for example, don't do this: `CT.A_1.fastq`). Dots in the filename will cause problems later in the analysis.

7. How many sequences do you get in total?

8. What is the average read length?

Check that you have 14 different files, two per colony. Check the sizes of the files, if the ones containing forward reads are different from the ones containing the reverse reads, if the correct samples are in the correct colonies. You can use:

`ls -l`, `wc -l` and `md5sum`

for this (what does md5sum do? Do you want md5sum to be the same or different for these different files?).

1.2.2 Primers

The primer sequences that were used in the study, are not specified in the article, but a reference is given. It references a supplemental material found at:
http://www.nature.com/ismej/journal/v6/n8/extref/ismej20128x2.txt

Can you access the file? This is the paper that should have been referenced https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3063599/. Here, we find the sequencing primers for the V4 hyper-variable region:

Forward primer
`GTGCCAGCMGCCGCGGTAA`

Reverse primer
`GGACTACHVGGGTWTCTAA`

We have to check if the primers have been removed or not (we just use the first part of the primer):

```
cat *.fastq | paste - - - - | cut -f 2 | grep "^GTGCCAGC"
cat *.fastq | paste - - - - | cut -f 2 | grep -P "^TTAGA[CG]AC"
```

9. What is the meaning of `- - - -` in the `paste` command?

10. What are we doing in the second line?

We don't seem to have any primers to remove.

1.3 Installation

Now we have to install the software that we will use in the analysis:

Already installed: FastQC
Checks the quality of sequence reads. Web:
https://www.bioinformatics.babraham.ac.uk/projects/fastqc/. Read the file 'Making sense of FastQC' to understand how this program works.

Already installed: Trimmomatic
Trims and discards sequence reads. Web:
http://www.usadellab.org/cms/?page=trimmomatic

pandaseq
This software can either be run on the server or compiled locally following instructions at https://github.com/neufeld/pandaseq. `pandaseq` merges forward and reverse reads based on the overlap. You can install it using Conda to overcome it asking for an admin password. Web:
https://github.com/neufeld/pandaseq/wiki/PANDAseq-Assembler

Vsearch
Software for performing various tasks for amplicon sequences such as clustering and chimera filtering. Follow the compilation instructions. You can skip the final step `make install` (which requires `sudo` access). The binary is created at the path `bin/vsearch` within the Vsearch directory https://github.com/torognes/vsearch

Normal procedure:

```
git clone <path to repo>
cd <name of repo>
./autogen.sh
./configure
make
```

The binaries are produced within the repositories `pandaseq` and `bin/vsearch`.

2 Analysis

2.1 Quality control

Before you begin this chapter, read the document FastQC_Manual.pdf.

As always, we have to check our sequences. We do this as usual by running `fastqc`. Create a folder called `1_fastqc` and use a `bash while` loop to generate FastQC reports for each sample. Store these in the `1_fastqc` folder. We will only take a look at sample `CT_A` (two files). In a real study each `fastqc` report has to be analyzed. In particular, look at the positional quality values (top graph). Forward reads seem to be fine, but in the reverse reads we have a drop of quality values towards the end. However, we will not need to trim the sequences. Remember that the reads will be merged in the 3 prime ends, like:

Forward 5'————-xxxxx3' Reverse 3'xxxxx————-5'

This means that the low-quality end of the reversed read will be aligned to, in our case, the high-quality end of the forward reads. `Trimmomatic` normally takes care of aligning the overlap by also considering the quality values. If you are interested, this is described at: http://microbiomejournal.biomedcentral.com/articles/10.1186/2049-2618-2-6.

11. Study the graphs. You can compare them to this example report https://www.bioinformatics.babraham.ac.uk/projects/fastqc/? What is the meaning of two graphs.

12. Some quality measures are expected to give warnings for our data. Can you explain why?

2.2 Sequence clipping and removal

Next, we perform quality trimming of the reads using the software Trimmomatic. Trimmomatic can both discard low-quality reads and trim low-quality ends of otherwise high-quality reads.

`trimmomatic` is a `java` program. They are often run like:

```
java -jar file.jar
```

Store it in a suitable `bin` directory. Next, create a directory called `2_trimming` in your project folder. Try the first run with sample CT_A. You may need to change the path to the jar file. A common error in this part has the wrong folder structure. Check yours before running the program. You should type the command in a single line without the backslashes.

```
java -jar ~/bin/Trimmomatic/trimmomatic-0.39.jar PE \
    -threads 4 \
    -trimlog CT_A.log \
    -baseout 2_trimming/CT_A.fastq \
    Raw/Colonies/CT_A_1.fastq \
```

```
    Raw/Colonies/CT_A_2.fastq \
    LEADING:20 TRAILING:20 SLIDINGWINDOW:8:15 \
    MINLEN:140 AVGQUAL:20
```

PE
Specifies that this is paired-end sequencing.

-threads 4
Run with four threads.

-trimlog logfile
Write log to file logfile.

-baseout basename
Give output file basename "basename".

LEADING:20
Cut bases off the start of a read, if below a threshold quality (here 20).

TRAILING:20
Cut bases off the end of a read, if below a threshold quality (here 20).

SLIDINGWINDOW:8:15
Performs a sliding window trimming approach. It starts scanning at the 5' end and clips the read once the average quality falls below a threshold. Here, we clip the read if the average quality is less than 15 in a window of 8 nucleotides.

MINLEN:140
Drop the read if it is below 140 nucleotides.

AVGQUAL:20
Drop the read if the average quality is below the specified level.

13. What is the meaning of a backslash at the end of a line in the provided Trimmomatic code?

Take a look at the files:

```
ls -l
wc -l *fastq
```

Note that `1U` contains forward sequences that were discarded or that were not found in the final reverse read file `CT_A_2P.fastq` (and vice versa for `2U`). Also note that the files to be used later on, `CT_A_1P.fastq` and `CT_A_2P.fastq`, contain equal number of sequences (as they should). There are many other parameters, one of them is the ability to remove Illumina adapters (you can see the adapters fasta files in the `adapters` directory found within the folder containing the Trimmomatic software). Instead of running programs over and over again on different data, automate this with the `bash` `while` loop. This is more efficient and reduces the risk of making mistakes.

```
ls ../Colonies/*_1.fastq | sed s/_1.fastq// | while read line; do id=$(echo $
line | sed "s/.*\///"); java -jar ~/bin/Trimmomatic-0.39/trimmomatic-0.39.jar
PE -threads 4 -trimlog $id.log -baseout $id.fastq ${line}_1.fastq ${line}_2.f
astq LEADING:20 TRAILING:20 SLIDINGWINDOW:8:15 MINLEN:140 AVGQUAL:20; done
```

Analyze the command. `${line}` means that the variable name is `line`. Since underscore is a valid character in a variable name, `$line_1.fastq` would contain the variable name `line_1`. It is generally preferable to use full variable syntax, including the curly braces.

14. What percentage of the reads was discarded?

15. Run the FastQC command again for the trimmed output. Choose three different categories in the fastqc report and compare the graphs the program generated. Discuss how they differ before and after the trimming.

Now we should have high-quality `fastq` files to merge. Create **the same directory structure** which you have locally on the server. Move the produced (paired) files to the server. Compress the files before transferring.

2.3 Merging forward and reverse reads

We run `Pandaseq` on the server (or locally if you compiled it yourself). Create the directory `3_mergereads` in your `Amplicon` directory and move there. Here, is an example of how to run `pandaseq`:

```
pandaseq \
    -f 2_trimming/CT_A_1P.fastq \
    -r 2_trimming/CT_A_2P.fastq \
    -g 3_mergereads/CT_A.log \
    -u 3_mergereads/CT_A.unaligned.fasta \
    -w 3_mergereads/CT_A.fastq \
    -F
```

Explanation of parameters:

-f
forward fastq file

-r
reverse fastq file

-g
log file

-u
unaligned sequences, fasta

-w
merged sequences, fastq

-F
output format fastq

Run for all other colonies as well. Automate with a `bash` `while` or `for` loop. If running on the server, copy your `3_mergereads` directory to your local computer (use `scp -r`).

16. What does `pandaseq` do?

17. What is the average length of the merged sequences?

2.4 Dereplicate, merge and cluster reads

Before we do the chimera checking we will do the following preprocessing of the data, all using the open source software Vsearch (which is a clone of the well known closed-source software Usearch). There are different opinions on which is the best way to perform this part of the analysis. We will be following a workflow recommended by Vsearch (https://github.com/torognes/vsearch/wiki/VSEARCH-pipeline, check out the right box as well)

18. Describe the rationale of the Vsearch recommended workflow.

First, we will preprocess the reads in preparation for the chimera checking. Here, we:

- Perform preliminary dereplication of reads within samples while relabelling each read to specify to which sample it belongs.
- Pool reads from all samples into a single file.
- Dereplicate reads across samples by removing duplicate sequences.
- Precluster reads - create groups of similar sequences across all samples.

2.4.1 Preliminary dereplication of samples

Create a new folder called `4_precluster`. We perform an initial dereplication of the samples to speed up subsequent steps and label reads with to which sample they belong.

```
vsearch \
    --fastx_uniques 3_mergereads/CT_A.fastq \
    --strand plus \
    --sizeout \
    --relabel CT_A \
    --fasta_width 0 \
    --fastaout 4_precluster/CT_A.derep.fasta
```

Meaning of each flag:

–derep_fulllength
Specify that we are doing dereplication and the input to dereplicate.

–strand
Specify which strand is expected in the input data. This is often crucial to handle correctly during analysis. **Analyzing stranded samples with wrong settings may lead to significant loss of information**. Strand refers to whether the reads belong to the forward or reverse DNA strand (https://en.wikipedia.org/wiki/Coding_strand). Whether strand-information is available depends on the sequencing protocol.

–sizeout
Write abundance information in the FASTA header output.

–fasta_width 0
Specify that we have single-line FASTA format.

–output filename
The resulting dereplicated sequences.

19. What is the purpose of the initial dereplication step? In which situation is it most relevant to do this?

20. Investigate the resulting file using `less`. What does `–relabel` do? What happened to the headers?

21. What percentage of the reads did you discard during dereplication (not how many your kept)? Make sure you understand how the reads are represented before and after dereplication.

Verify that the same number of sequences are present in the dereplicated file and in the input file. You can achieve this with a set of bash commands extracting and adding the 'size' part of the headers.
Hint: One easy way to sum a column of numbers in Bash using awk:

```
awk '{sum += $1} END {print sum}'
```

Perform dereplication for each of your files using a bash loop.

2.4.2 Pooling of reads

Now, we will pool all the reads into a single file. The samples could also be processed separately, but pooling them allows us to perform chimera checking and clustering across samples which speeds up and potentially improves the analysis.

Use the `cat` command to produce a single file containing all the dereplicated FASTA-sequences called `all_combined.fasta`

22. Verify that the resulting file contains the same number of reads and nucleotides as the original files added together. How many reads are in the resulting file?

2.4.3 Dereplicate reads

Next, we proceed with dereplicating the full set of reads. We will need to map back reads to samples later. We therefore include the –uc flag in the output which contains information about how the reads were combined. Furthermore, we need to specify the –sizein flag as we now have abundance information in each FASTA header.

```
vsearch \
    --derep_fulllength 4_precluster/all_combined.fasta \
    --sizein \
    --sizeout \
    --fasta_width 0 \
    --uc 4_precluster/all_derep.uc \
    --output 4_precluster/all_derep.fasta
```

23. What is the purpose of this dereplication step?

24. How many reads does the most frequent sequence from the Vsearch output represent?

25. Calculate the number of reads from each sample. This can be achieved by looping over the sample names and extracting corresponding size counts for each. Make sure they correspond to the previous counts. Write down the number of reads per sample.

2.4.4 Preclustering

Next, we will do a preliminary clustering (pre-clustering) of the reads in preparation for the chimera checking. This will greatly reduce the processing time in subsequent steps while having a tradeoff of losing some sensitivity. We use a threshold of 0.97, meaning that sequences at least 97% similar will be grouped together.

```
vsearch \
    --cluster_size 4_precluster/all_derep.fasta \
    --id 0.97 \
    --strand plus \
    --sizein \
    --sizeout \
    --fasta_width 0 \
    --uc 4_precluster/all_preclust.uc \
    --centroids 4_precluster/all_preclust.fasta
```

The centroids option produces a FASTA file with the so-called centroid sequences, which are sequences that are selected as representatives for each cluster. The full information about the preclustering step is retained in the all_preclust.uc file. This will be needed later to backtrack the reads after performing chimera checking.

26. What is the purpose of the preclustering step?

27. How many FASTA entries do we have after preclustering?

We produced `uc` files in both dereplication and preclustering. This file provides information on which clustered sequences belong together. Study it further at https://www.drive5.com/usearch/manual/opt_uc.html. Next, count the number of different record types in the first column. Make sure that the number of centroids corresponds to the number of sequences in the `all_preclust.fasta` file.

28. What is the number of centroids in the file?

2.5 Chimera checking

Finally, we are ready for chimera checking. Start by reading more about chimeras at https://www.drive5.com/usearch/manual/chimeras.html. Chimeric sequences are artifacts produced during PCR amplification, which combines parts of two real reads into an artificial one. These can be very hard to accurately distinguish from the real ones. There are two ways of performing chimera checking:

- *De novo*, which means that the sequences in the file are compared to each other.
- Reference-based, where sequences are compared to a database that contains chimera free sequences.

Here, we will first perform *de novo* based chimera checking, followed by reference-based chimera checking. Chimeras are very difficult to be accurately identified, but the software for detecting them have been continuously developed during the last years (this section of the course material needed to be updated each year for several years now due to changing software).

29. What would have happened if we had run first the reference step and then the *de novo*?

2.5.1 *De novo* chimera checking

First, we perform *de novo* based chimera checking. Start by creating a directory called `5_chimera` on your local computer. Next, we run the Vsearch command for *de novo* chimera checking:

```
vsearch \
    --uchime3_denovo 4_precluster/all_preclust.fasta \
    --threads 4 \ # Depends on your computer
    --sizein \
    --sizeout \
    --fasta_width 0 \
    --nonchimeras 5_chimera/all.denovo.nonchimeras.fasta \
    --chimeras 5_chimera/all.denovo.chimeras.fasta \
    --uchimeout 5_chimera/all.denovo.uchime
```

30. Make sure all three files were created. Describe the content of each file.

31. What does *de novo* actually mean, and why do we call this kind of chimera checking *de novo*?

32. What percentage of the reads was classified as chimeric?

Make sure you understand the purpose of the different options. This step could be performed directly on raw sequences, but would take much more time.

2.5.2 Reference-based chimera checking

Next, we perform reference-based chimera checking. For this, we need to download a reference database. We recommend you use the following: https://mothur.s3.us-east-2.amazonaws.com/wiki/silva.gold.bacteria.zip. Retrieve it and place it for instance in your `data` directory. It is given as an alignment file, so make sure to trim away '-' and '.' characters. It can be done as following:

```
unzip -p Silva.gold.bacteria.zip silva.gold.align | \
    sed -e "s/[.-]//g" > gold.fasta
```

33. Inspect the database. How many sequences does it contain? Approximately how long are the sequences?

Now we are ready to perform the reference-based checking:

```
vsearch \
    --uchime_ref 5_chimera/all.denovo.nonchimeras.fasta\
    --db ../Raw/gold.fasta
    --sizein \
    --sizeout \
    --fasta_width 0 \
    --nonchimeras 5_chimera/all.ref.nonchimeras.fasta \
    --chimeras 5_chimera/all.ref.chimeras.fasta \
    --uchimeout 5_chimera/all.ref.uchime
```

Take a look at the output report (you can import it into a spreadsheet):

```
less all.ref.uchime
```

The columns are explained at:
http://drive5.com/uchime/uchime_quickref.pdf.

34. What does each column represent?

Check if the numbers are correct (for both chimera checking steps):

```
# Find out the column number for chimeric (Y) or not (N)
cat all.ref.uchime | head -1 | tr "\t" "\n" | cat -n
cat all.ref.uchime | cut -f 18 | sort | uniq -c
cat all.ref.chimeras.fasta | grep -c \>
cat all.ref.nonchimeras.fasta | grep -c \>
```

35. How many reads were discarded (in percentage) in each of the chimera checking steps?

Now we have high-quality reads. Remember that we have reduced the number of reads in two steps:

- We first dereplicated reads, removing duplicate reads.
- Next, we did a preclustering step before doing our chimera checking.

But we are not ready to do the clustering step yet. Before we perform the final clustering, we want to obtain a set of all reads that were non-chimeric.

2.6 Clustering

2.6.1 Retrieve filtered original reads

The resulting precluster sequences have been used during chimera checking. We want to retrieve all reads that are part of preclusters classified as not chimeric and save them in a directory called `6_clustering`.

36. You can use the Perl script `map.pl` available on the course web page. Use it in the following way:

```
# Extract all reads belonging to non-chimeric preclusters
# all_derep.fasta: all reads dereplicated across colonies
# all_preclust.uc: information about the clustering step
# all.ref.nonchimeras.fasta: preclusters classified
#   as non chimeric
map.pl \
    4_precluster/all_derep.fasta \
    4_precluster/all_preclust.uc \
    5_chimera/all.ref.nonchimeras.fasta > \
    6_clustering/all.nonchimeras.derep.fasta

# Next, we extract all non-chimeric reads
# all_combined.fasta: the starting set of reads
# all_derep.uc: information about the dereplication step
# all.nonchimeras.derep.fasta: dereplicated sequences
#   classified as non-chimeric (from previous step)
map.pl \
    4_precluster/all_combined.fasta \
    4_precluster/all_derep.uc \
    6_clustering/all.nonchimeras.derep.fasta > \
    6_clustering/all.nonchimeras.fasta
```

39. How many non-chimeric preclusters, dereplicated reads and raw reads do we have?

40. For each step, compare the percentage of the total number of reads, dereplicated reads and preclusters that are classified as chimeric.

As a last step, we need to rename the FASTA headers in the final output for the subsequent programs to see which sequences belong to the same sample.

```
sed "s/[0-9]\+;/;/" 6_clustering/all.nonchimeras.fasta > 6_clustering/all.non
chimeras.renamed.fasta
```

41. Study Linux *sed* command. How does it change the sample headers?

2.6.2 Perform clustering

Finally, we are ready for the final clustering of the clean reads. The purpose is to reduce the amount of data that has to be analyzed. Many software can't handle millions of reads (like GreenGenes). For bacteria we normally group sequences that are less than 3% divergent into the same cluster. This could be called a proxy for a bacterial species (also called 'operational taxonomical unit', or 'OTU'). Note that we are using a different clustering threshold now as the purpose of the clustering is different.

```
vsearch \
    --cluster_size 6_clustering/all.nonchimeras.renamed.fasta \
    --threads 4 \ # Depends on your computer
    --id 0.97 \
    --strand plus \
    --sizein \
    --sizeout \
    --fasta_width 0 \
    --relabel OTU_ \
    --mintsize 3 \
    --uc 6_clustering/final.uchime \
    --centroids 6_clustering/otus.fasta \
    --otutabout 6_clustering/otus.tsv
```

–cluster_size
A suitable clustering algorithm for 16S rRNA.

–mintsize 3
We only output clusters that have three or more reads. This is the way it was done in the article.

Investigate the resulting files. The `otus.fasta` is an OTU table containing information on how many reads from each sample are present in each OTU. This file can be used together with taxonomical classifications for further downstream analysis.

42. Calculate the total number of OTUs and how many sequences they represent together. Does this correspond to the number of input sequences?

43. How many OTUs were identified in each of the seven colonies? (Hint: investigate the `otus.tsv` file)

44. What was the average number of sequences per OTU in each of the seven colonies?

2.7 Classifying

Finally, we assign a taxonomical group to the samples based on the centroid sequences for the OTUs. Create a directory called `7_classify`. For this, we use a software called RDP classifier. This classification can be done directly on the RDP web site, but with larger data sets this quickly become unmanageable. The classifier understands both the `fasta` and `fastq` formats. Download the classifier from:
http://sourceforge.net/projects/rdp-classifier/
If it takes too long to download, ask the teacher for help.

The download consists of the program as such plus a database of 16S rRNA sequences that have been manually curated. The query sequences are compared to that of the database. It's possible (but tricky) to make your own database.

45. Explain what the RDP database is.

46. How many sequences are in it?

47. Explain how can the RDP database increase or decrease in size in future releases.

If you have installed the classifier in `bin`, it can be started like:

```
java -Xmx1g -jar ~/bin/rdp_classifier_2.12/dist/classifier.jar
```

But to make it easier to use the classifier you can add the following to the `.bashrc` file (make sure to double check your version number):

```
alias rdp='java -Xmx1g -jar ~/bin/rdp_classifier_2.12/dist/classifier.jar'
# Don't forget to source
```

Then you start it as the following. NOTE: If you are running this workflow in Snakemake it will likely be unable to recognize the alias. In that case we recommend you type out the full command.

```
rdp classify \
    -c 0.8 \
    -f fixrank \
    -o 7_classify/all.fixedRank \
    -h 7_classify/all.significant \
    6_clustering/otus.fasta
```

- Java options
  - `-Xmx1g` means that a maximum memory of 1 GB can be used.
  - `-jar filename` means that java should read in the program/package found in the filename.

For an explanation of the classifier specific options:

```
rdp -h
```

As you see you didn't have to specify `-c 0.8` since it is the default. But it's often useful to also specify the defaults. Then you understand directly from your documentation what the defaults are instead of looking them up with `-h`. Here are the options explained again:

- RDP Classifier options:
  - `classify` means that we want to classify the sequences. There are also other options like `libcompare` which compares two different samples.
  - `-c` The assignment confidence cutoff (default is 0.8, but it's always good to specify the defaults).
  - `-f fixrank` only outputs domain, phylum, class, order, family and genus together with confidence levels.
  - `-o` tab-delimited output file containing the classification for each sequence.
  - `-h` an output file containing the assignment count for each taxon for the specified confidence level.

There are three output files: `all.fixedRank`, `all.significant` and `cnadjusted_all.significant`. Take a look at all three. The first file contains the classification for each sequence. Study the file further. The file is tab-delimited and when there are many columns it's often useful to get a list of the fields:

```
head -1 all.fixedRank | tr "\t" "\n" | cat -n
```

48. How many cluster sequences could
    1. *not* be classified as bacterial?
    2. be determined on the phylum level (on the 0.8 confidence level)?

Note that each sequence represents many reads according to the `size` tag. To see the phylum distribution, the following code could have been used:

```
cat all.fixedRank | cut -f 9 | sort -n | uniq -c
```

If you just look with `less all.fixedRank` it would seem that the values to be used should be in column 8. But test:

```
head all.fixedRank | tr "\t" @
```

If you are an `awk` lover (who isn't?):

```
awk -F "\t" ' {if ($8>=0.8) ++i} END {print i} ' all.fixedRank
```

To doublecheck the number, try the other output file:

```
cat all.significant | grep -P "\tphylum\t" | sed "s/.*\t//" | tr "\n" + | sed "s/+$/\n/" | bc
```

Evaluate the command.

49. How many cluster sequences were determined on the genus level (using 0.8)?

50. How would it change if you had used thresholds of 0.7 and 0.9?

The results here are quite normal. Many published 16S rRNA studies only report the results on genus level and above. The number of encountered OTUs (taxa) is often low.

Finally, we will do some post-processing of the final results. Normally a significant part of the analysis will be here. Producing the OTU table and taxonomy is just the starting point.

Link the phylum for each OTU to the read counts in the OTU table obtained after clustering. Only include cluster sequences that are bacterial and that were mapped with a confidence level of 0.8 and above.

51. Plot your results with R and compare it with the article's figure 2. Discuss the differences between plots.

These are all the mandatory questions that you have to answer. If you have some extra time, take a look at the following questions.

Challenge exercises

53. MultiQC (https://multiqc.info/) is a powerful software that can create a combined report for many types of output, including FastQC. Take a look and generate combined reports of FastQC files before and after the trimming step.

54. A commonly performed analysis is calculating so-called alpha- and beta-diversity indices for the bacterial communities. This is measures of how much heterogeneity there is within (alpha) and between (beta) samples. Follow the tutorial at http://scikit-bio.org/docs/0.4.1/diversity.html to calculate and visualize indices and principal component analysis of the samples using the `skbio.diversity` Python package.

55. Investigate the copy-number adjusted cluster file (cnadjusted_all.significant). See if you can use this information to adjust the OTU table numbers for copy number variations and plot the results.

56. Rerun the calculations above using the commercial software Usearch (https://www.drive5.com/usearch). Vsearch is developed to closely mimic Usearch and the processing steps should be the same, but vary some in the algorithms as Usearch is closed-source. Compare the result and see if there is (1) any significant difference or (2) any difference.

57. (Extra optional). Put together an R-shiny application for interactively investigating the above data. Options could include what taxa level to view at, whether copy-number alterations should be taken into account and whether the data was processed in Vsearch or Usearch. Rshiny is a framework for easily making interactive R-plots. It is less intimidating than it sounds, and is a skill currently in great demand. Here is a potential starting point: https://shiny.rstudio.com/tutorial

When you finished, upload your final Word document to Canvas.

2.8 Other tools

Due to the complexity of these procedures, pipelines such as the one you have worked on have been developed by other groups. The two most well known are `QIIME` and `mothur`:
http://qiime.org/
http://www.mothur.org/
They incorporate many of the programs used in this exercise. Learning how to use these pipelines takes a couple of days. Also, since input from one program is dependent on the output from the previous, the pipelines are restricted to certain (older) versions of the programs as well as of python modules (true for `QIIME`).