Computational Genomics Tutorial

Release 2020.2.0

Sebastian Schmeier (https://sschmeier.com)

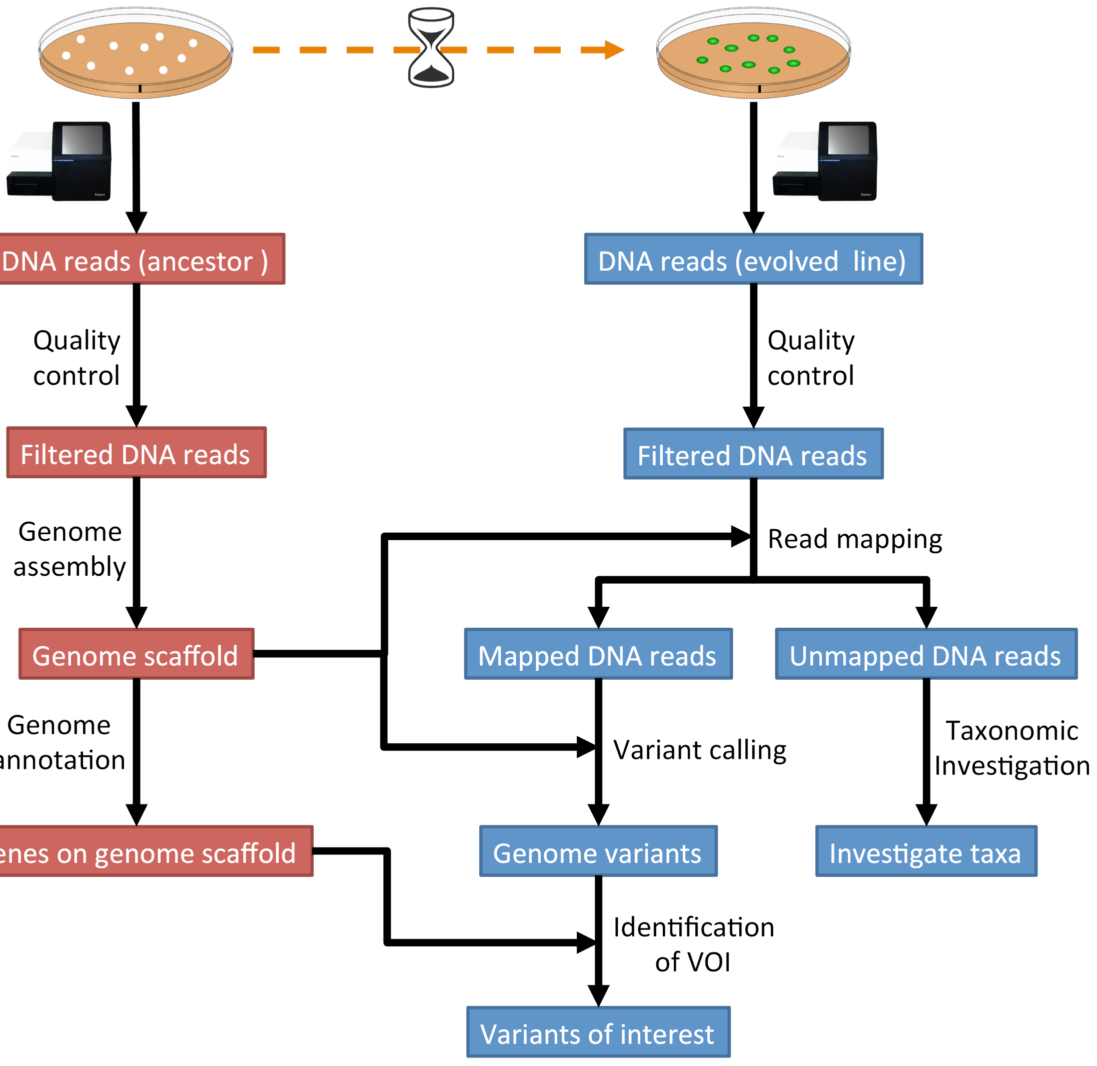
Jun 01, 2020

CONTENTS

Attention: This is a **new revised release**, now using data from **E. coli** to speed the analysis steps up. The former tutorial based on fungi data can be accessed at https://genomics-fungi.sschmeier.com/.

This is an introductory tutorial for learning computational genomics mostly on the Linux command-line. You will learn how to analyse next-generation sequencing (NGS) data. The data you will be using is real research data. The final aim is to identify genome variations in evolved lines of wild yeast that can explain the observed biological phenotypes. Until 2020, Sebastian[1] was teaching this material in the Massey University course Genome Science[2].

More information about other bioinformatics material and our past research can be found on the former webpages of the Schmeier Group[3] (https://www.schmeierlab.com).



Note: A online version of this tutorial can be accessed at https://genomics.sschmeier.com.

[1] https://www.sschmeier.com
[2] https://www.massey.ac.nz/massey/learning/programme-course/course.cfm?course_code=203341
[3] https://www.schmeierlab.com

CHAPTER

ONE

INTRODUCTION

This is an introductory tutorial for learning genomics mostly on the Linux command-line. Should you need to refresh your knowledge about either Linux or the command-line, have a look here[4].

In this tutorial you will learn how to analyse next-generation sequencing (NGS) data. The data you will be using is actual research data. The experiment follows a similar strategy as in what is called an "experimental evolution" experiment [KAWECKI2012], [ZEYL2006]. The final aim is to identify the genome variations in evolved lines of *E. coli* that can explain the observed biological phenotype(s).

1.1 The workflow

The tutorial workflow is summarised in Fig. 1.1.

1.2 Learning outcomes

During this tutorial you will learn to:

- Check the data quality of an NGS experiment
- Create a genome assembly of the ancestor based on NGS data
- Map NGS reads of evolved lines to the created ancestral reference genome
- Call genome variations/mutations in the evolved lines
- Annotate a newly derived reference genome
- Find variants of interest that may be responsible for the observed evolved phenotype(s)

[4] http://linux.sschmeier.com/

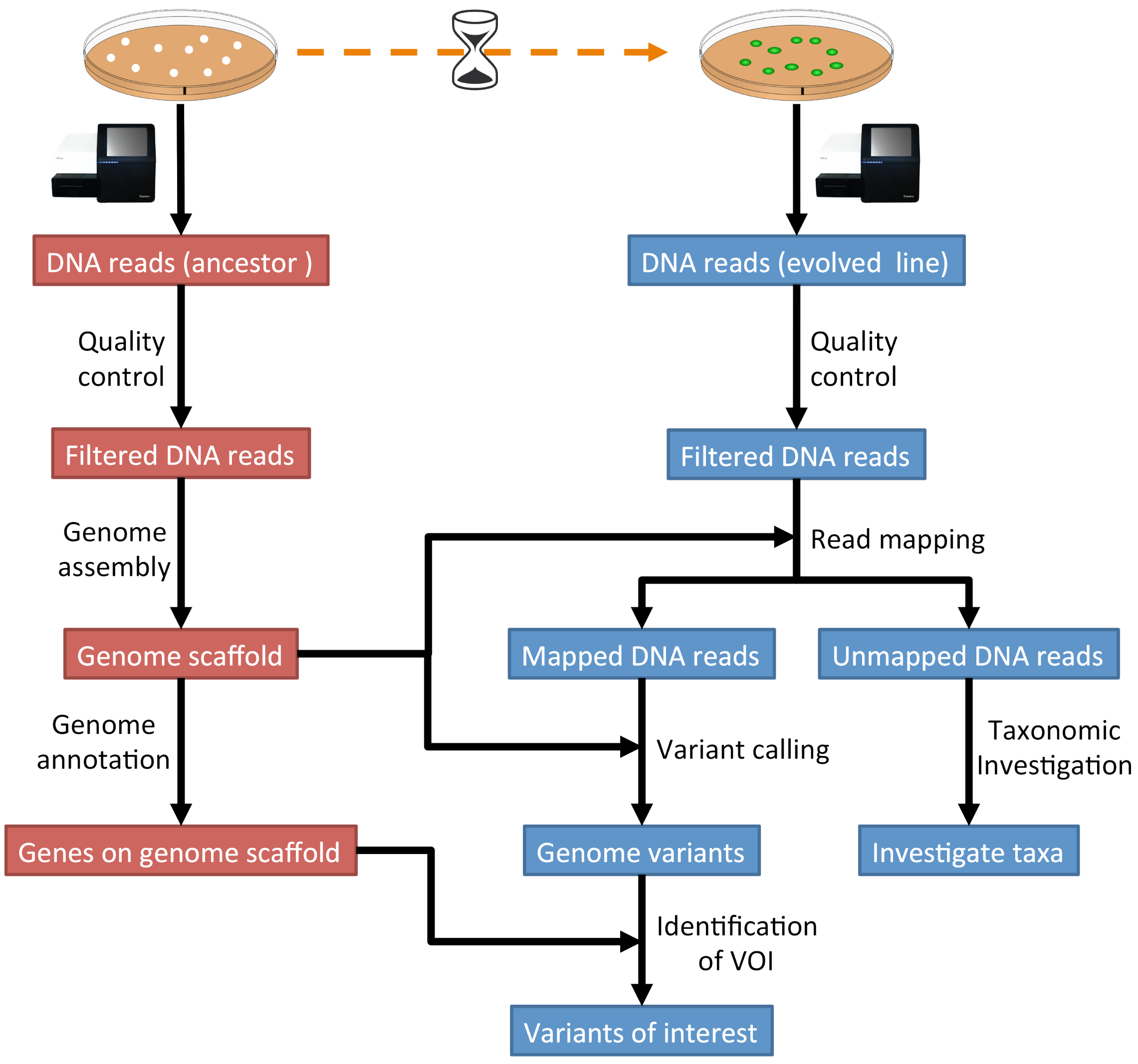


Fig. 1.1: The tutorial will follow this workflow.

CHAPTER

TWO

TOOL INSTALLATION

2.1 Install the conda package manager

We will use the package/tool managing system conda[7] to install some programs that we will use during the course. It is not installed by default, thus we need to install it first to be able to use it. Let us download conda[8] first:

```
# download latest conda installer
$ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Note: Should the conda installer download fail. Please find links to alternative locations on the *Downloads* (page 77) page.

Now lets install conda[9]:

```
# run the installer
$ bash Miniconda3-latest-Linux-x86_64.sh
```

After you accepted the license agreement conda[10] will be installed. At the end of the installation you will encounter the following:

```
...
installation finished.
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>>
```

Please type **"yes"** here. This will add some code to your *.bashrc* init file, which is important to work with conda[11] correctly.

Attention: Please close and reopen the terminal, to complete the installation.

After closing and re-opening the shell/terminal, we should be able to use the conda[12] command:

```
$ conda update --yes conda
```

[7] http://conda.pydata.org/miniconda.html
[8] http://conda.pydata.org/miniconda.html
[9] http://conda.pydata.org/miniconda.html
[10] http://conda.pydata.org/miniconda.html
[11] http://conda.pydata.org/miniconda.html
[12] http://conda.pydata.org/miniconda.html

2.1.1 Installing conda channels to make tools available

Different tools are packaged in what conda[13] calls channels. We need to add some channels to make the bioinformatics and genomics tools available for installation:

```
# Install some conda channels
# A channel is where conda looks for packages
$ conda config --add channels defaults
$ conda config --add channels bioconda
$ conda config --add channels conda-forge
```

Attention: The order of adding channels is important. Make sure you use the shown order of commands.

2.2 Creating environments

We create a conda[14] environment for some tools. This is useful to work **reproducible** as we can easily re-create the tool-set with the same version numbers later on.

```
$ conda create -n ngs python=3
# activate the environment
$ conda activate ngs
```

So what is happening when you type `conda activate ngs` in a shell. The `PATH` variable of your shell gets temporarily manipulated and set to:

```
$ echo $PATH
/home/guest/miniconda3/bin:/home/guest/miniconda3/condabin:...
$ conda activate ngs
$ echo $PATH
/home/guest/miniconda3/envs/ngs/bin:/home/guest/miniconda3/condabin: ...
```

Now it will look first in your environment's bin directory but afterwards in the general conda bin (`/home/guest/miniconda3/condabin`). So basically everything you install generally with conda (without being in an environment) is also available to you but gets overshadowed if a similar program is in `/home/guest/miniconda3/envs/ngs/bin` and you are in the `ngs` environment.

2.3 Install software

To install software into the activated environment, one uses the command `conda install`.

```
# install more tools into the environment
$ conda install package
```

Note: To tell if you are in the correct conda environment, look at the command-prompt. Do you see the name of the environment in round brackets at the very beginning of the prompt, e.g. (ngs)? If not, activate the `ngs` environment with `conda activate ngs` before installing the tools.

[13] http://conda.pydata.org/miniconda.html
[14] http://conda.pydata.org/miniconda.html

2.4 General conda commands

```
# to search for packages
$ conda search [package]

# To update all packages
$ conda update --all --yes

# List all packages installed
$ conda list [-n env]

# conda list environments
$ conda env list

# create new env
$ conda create -n [name] package [package] ...

# activate env
$ conda activate [name]

# deactivate env
$ conda deactivate
```

CHAPTER

THREE

QUALITY CONTROL

3.1 Preface

There are many sources of errors that can influence the quality of your sequencing run [ROBASKY2014]. In this quality control section we will use our skill on the command-line interface to deal with the task of investigating the quality and cleaning sequencing data [KIRCHNER2014].

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

3.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 3.1.

3.3 Learning outcomes

After studying this tutorial you should be able to:

1. Describe the steps involved in pre-processing/cleaning sequencing data.
2. Distinguish between a good and a bad sequencing run.
3. Compute, investigate and evaluate the quality of sequence data from a sequencing experiment.

3.4 The data

First, we are going to download the data we will analyse. Open a shell/terminal.

```
# create a directory you work in
$ mkdir analysis

# change into the directory
$ cd analysis

# download the data
$ wget -O data.tar.gz https://osf.io/2jc4a/download

# uncompress it
$ tar -xvzf data.tar.gz
```

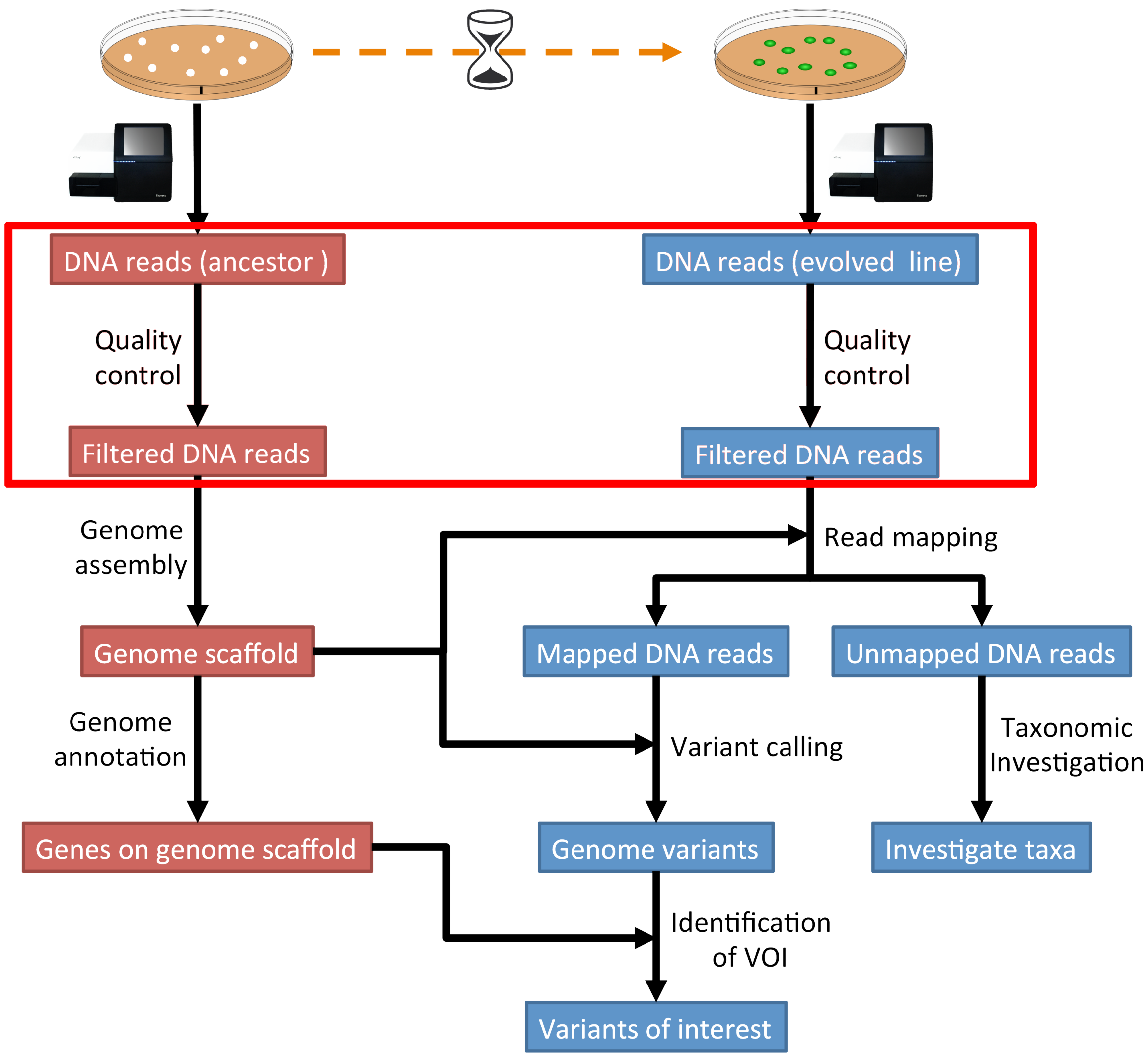


Fig. 3.1: The part of the workflow we will work on in this section marked in red.

Note: Should the download fail, download manually from *Downloads* (page 77). Download the file to the `~/analysis` directory and decompress.

The data is from a paired-end sequencing run data (see Fig. 3.2) from an Illumina[15] HiSeq [GLENN2011]. Thus, we have two files, one for each end of the read.

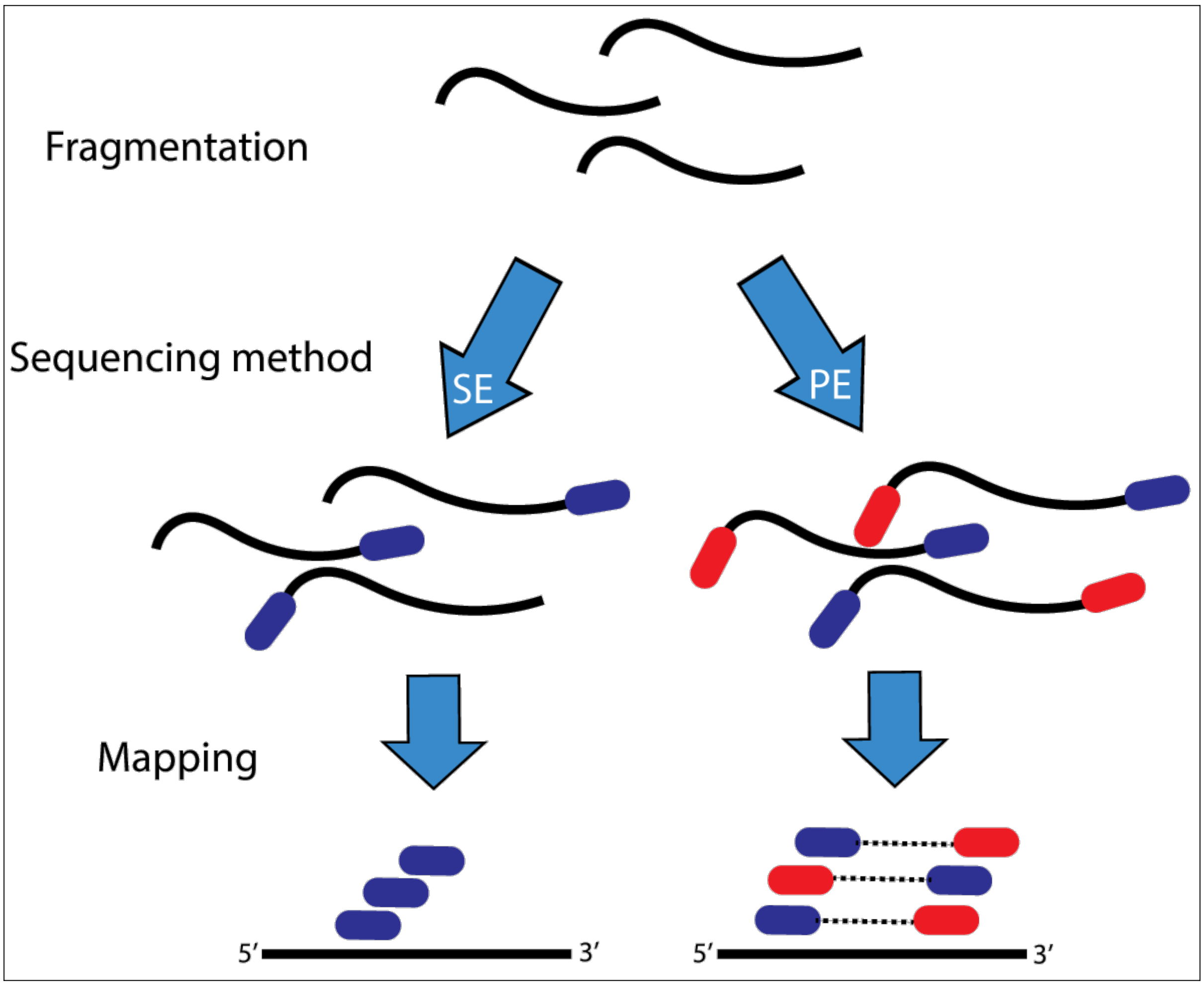


Fig. 3.2: Illustration of single-end (SE) versus paired-end (PE) sequencing.

If you need to refresh how Illumina[16] paired-end sequencing works have a look at the Illumina technology webpage[17] and this video[18].

Attention: The data we are using is "almost" raw data as it came from the machine. This data has been post-processed in two ways already. All sequences that were identified as belonging to the PhiX genome have been removed. This process requires some skills we will learn in later sections. Illumina[19] adapters have been removed as well already! The process is explained below and we are going to run through the process anyways.

[15] http://illumina.com
[16] http://illumina.com
[17] http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html
[18] https://youtu.be/HMyCqWhwB8E
[19] http://illumina.com

3.4.1 Investigate the data

Make use of your newly developed skills on the command-line to investigate the files in `data` folder.

Todo:

1. Use the command-line to get some ideas about the file.
2. What kind of files are we dealing with?
3. How many sequence reads are in the file?
4. Assume a genome size of ~4.6 MB. Calculate the coverage based on this formula: `C = LN / G`

- `C`: Coverage
- `G`: is the haploid genome length in bp
- `L`: is the read length in bp (e.g. 2x150 paired-end = 300)
- `N`: is the number of reads sequenced

3.5 The fastq file format

The data we receive from the sequencing is in `fastq` format. To remind us what this format entails, we can revisit the fastq wikipedia-page[20]!

A useful tool to decode base qualities can be found here[21].

Todo: Explain briefly what the quality value represents.

3.6 The QC process

There are a few steps one need to do when getting the raw sequencing data from the sequencing facility:

1. Remove PhiX sequences (we are not going to do this)
2. Adapter trimming
3. Quality trimming of reads
4. Quality assessment

3.7 PhiX genome

PhiX[22] is a nontailed bacteriophage with a single-stranded DNA and a genome with 5386 nucleotides. PhiX is used as a quality and calibration control for sequencing runs[23]. PhiX is often added at a low known concentration, spiked in the same lane along with the sample or used as a separate lane. As the concentration of the genome is known, one can calibrate the instruments. Thus, PhiX genomic sequences need to be removed before processing your data further as this constitutes a deliberate contamination [MUKHERJEE2015]. The steps involve mapping all reads to the "known" PhiX genome, and removing all of those sequence reads from the data.

[20] https://en.wikipedia.org/wiki/FASTQ_format
[21] http://broadinstitute.github.io/picard/explain-qualities.html
[22] https://en.wikipedia.org/wiki/Phi_X_174
[23] http://www.illumina.com/products/by-type/sequencing-kits/cluster-gen-sequencing-reagents/phix-control-v3.html

However, your sequencing provider might not have used PhiX, thus you need to read the protocol carefully, or just do this step in any case.

Attention: We are **not** going to do this step here, as the sequencing run we are using did not use PhiX. Please see the *Read mapping* (page 25) section on how to map reads against a reference genome.

3.8 Adapter trimming

The process of sequencing DNA via Illumina[24] technology requires the addition of some adapters to the sequences. These get sequenced as well and need to be removed as they are artificial and do not belong to the species we try to sequence. Generally speaking we have to deal with a trade-off between accuracy of adapter removal and speed of the process. Adapter trimming does take some time.

Also, we have generally two different approaches when trimming adapter:

1. We can use a tool that takes an adapter or list of adapters and removes these from each sequence read.
2. We can use a tool that predicts adapters and removes them from each sequence read.

For the first approach we need to know the adapter sequences that were used during the sequencing of our samples. Normally, you should ask your sequencing provider, who should be providing this information to you. Illumina[25] itself provides a document[26] that describes the adapters used for their different technologies. Also the FastQC[27] tool, we will be using later on, provides a collection of contaminants and adapters[28].

However, often (sadly) this information is not readily available, e.g. when dealing with public data. Thus, the second approach can be employed, that is, using a tool that predicts adapters.

Here, we are going to use the second approach with a tool called fastp to trim adapters **and** do quality trimming. fastp has a few characteristics which make it a great tool, most importantly: it is pretty fast, provides good information after the run, and can do quality trimming as well, thus saving us to use another tool to do this.

Quality trimming of our sequencing reads will remove bad quality called bases from our reads, which is especially important when dealing with variant identification.

```
# create env and install tools
$ conda create --yes -n qc fastp fastqc multiqc

# activate env
$ conda activate qc
```

Here, as an example we are trimming the sequence reads of the ancestor:

```
$ mkdir trimmed

$ fastp --detect_adapter_for_pe
        --overrepresentation_analysis
        --correction --cut_right --thread 2
        --html trimmed/anc.fastp.html --json trimmed/anc.fastp.json
        -i data/anc_R1.fastq.gz -I data/anc_R2.fastq.gz
        -o trimmed/anc_R1.fastq.gz -O trimmed/anc_R2.fastq.gz
```

[24] http://illumina.com
[25] http://illumina.com
[26] https://support.illumina.com/downloads/illumina-customer-sequence-letter.html
[27] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[28] https://github.com/csf-ngs/fastqc/blob/master/Contaminants/contaminant_list.txt

- `--detect_adapter_for_pe`: Specifies that we are dealing with paired-end data.
- `--overrepresentation_analysis`: Analyse the sequence collection for sequences that appear too often.
- `--correction`: Will try to correct bases based on an overlap analysis of read1 and read2.
- `--cut_right`: Will use quality trimming and scan the read from start to end in a window. If the quality in the window is below what is required, the window plus all sequence towards the end is discarded and the read is kept if its still long enough.
- `--thread`: Specify how many concurrent threads the process can use.
- `--html` and `--json`: We specify the location of some stat files.
- `-i data/anc_R1.fastq.gz -I data/anc_R2.fastq.gz`: Specifies the two input read files
- `-o trimmed/anc_R1.fastq.gz -O trimmed/anc_R2.fastq.gz`: Specifies the two desired output read files

Todo:

1. Run fastp also on the evolved samples.

Hint: Should you not get the commands together to trim the evolved samples, have a look at the coding solutions at *Code: fastp* (page 73). Should you be unable to run fastp at all to trim the data. You can download the trimmed dataset here[29]. Unarchive and uncompress the files with `tar -xvzf trimmed.tar.gz`.

3.9 Quality assessment of sequencing reads

3.9.1 Installing FastQC

```
$ fastqc --help
```

```
            FastQC - A high throughput sequence QC analysis tool

SYNOPSIS

        fastqc seqfile1 seqfile2 .. seqfileN

    fastqc [-o output dir] [--(no)extract] [-f fastq|bam|sam]
           [-c contaminant file] seqfile1 .. seqfileN

DESCRIPTION

    FastQC reads a set of sequence files and produces from each one a quality
    control report consisting of a number of different modules, each one of
    which will help to identify a different potential type of problem in your
    data.

    If no files to process are specified on the command line then the program
    will start as an interactive graphical application.  If files are provided
    on the command line then the program will run with no user interaction
    required.  In this mode it is suitable for inclusion into a standardised
    analysis pipeline.
```

[29] https://osf.io/m3wpr/download

3.9.2 FastQC manual

FastQC[30] is a very simple program to run that provides inforation about sequence read quality.

From the webpage:

> "FastQC aims to provide a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines. It provides a modular set of analyses which you can use to give a quick impression of whether your data has any problems of which you should be aware before doing any further analysis."

The basic command looks like:

```
$ fastqc -o RESULT-DIR INPUT-FILE.fq(.gz) ...
```

- `-o RESULT-DIR` is the directory where the result files will be written
- `INPUT-FILE.fq` is the sequence file to analyze, can be more than one file.

Hint: The result will be a HTML page per input file that can be opened in a web-browser.

Hint: The authors of FastQC[31] made some nice help pages explaining each of the plots and results you expect to see here[32].

3.9.3 MultiQC

MultiQC[33] is an excellent tool to put FastQC[34] (and other tool) results of different samples into context. It compiles all FastQC[35] results and fastp stats into one nice web-page.

The use of MultiQC[36] is simple. Just provide the command with a directories where multiple results are stored and it will compile a nice report, e.g.:

```
$ multiqc DIRECTORY DIRECTORY ...
```

3.10 Run FastQC and MultiQC on the trimmed data

Todo:

1. Create a directory for the results –> `trimmed-fastqc`
2. Run FastQC on all **trimmed** files.
3. Visit the FastQC[37] website and read about sequencing QC reports for good and bad Illumina[38] sequencing runs.
4. Run MultiQC[39] on the `trimmed-fastqc` and `trimmed` directories

[30] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[31] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[32] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/
[33] https://multiqc.info/
[34] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[35] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[36] https://multiqc.info/
[37] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[38] http://illumina.com
[39] https://multiqc.info/

5. Compare your results to these examples (Fig. 3.3 to Fig. 3.5) of a particularly bad run (taken from the FastQC[40] website) and write down your observations with regards to your data.
6. What elements in these example figures (Fig. 3.3 to Fig. 3.5) indicate that the example is from a bad run?

Hint: Should you not get it right, try the commands in *Code: FastQC* (page 73).

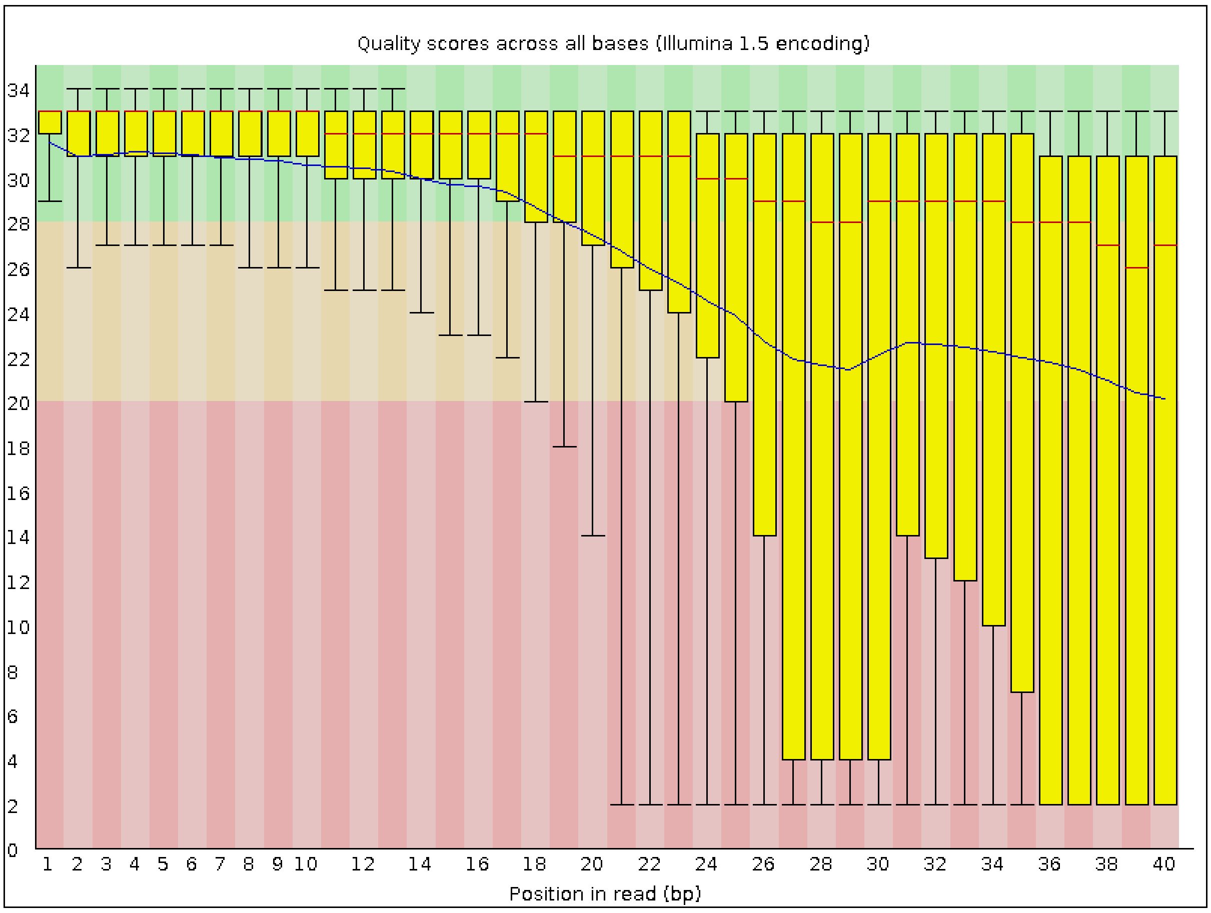


Fig. 3.3: Quality score across bases.

[40] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

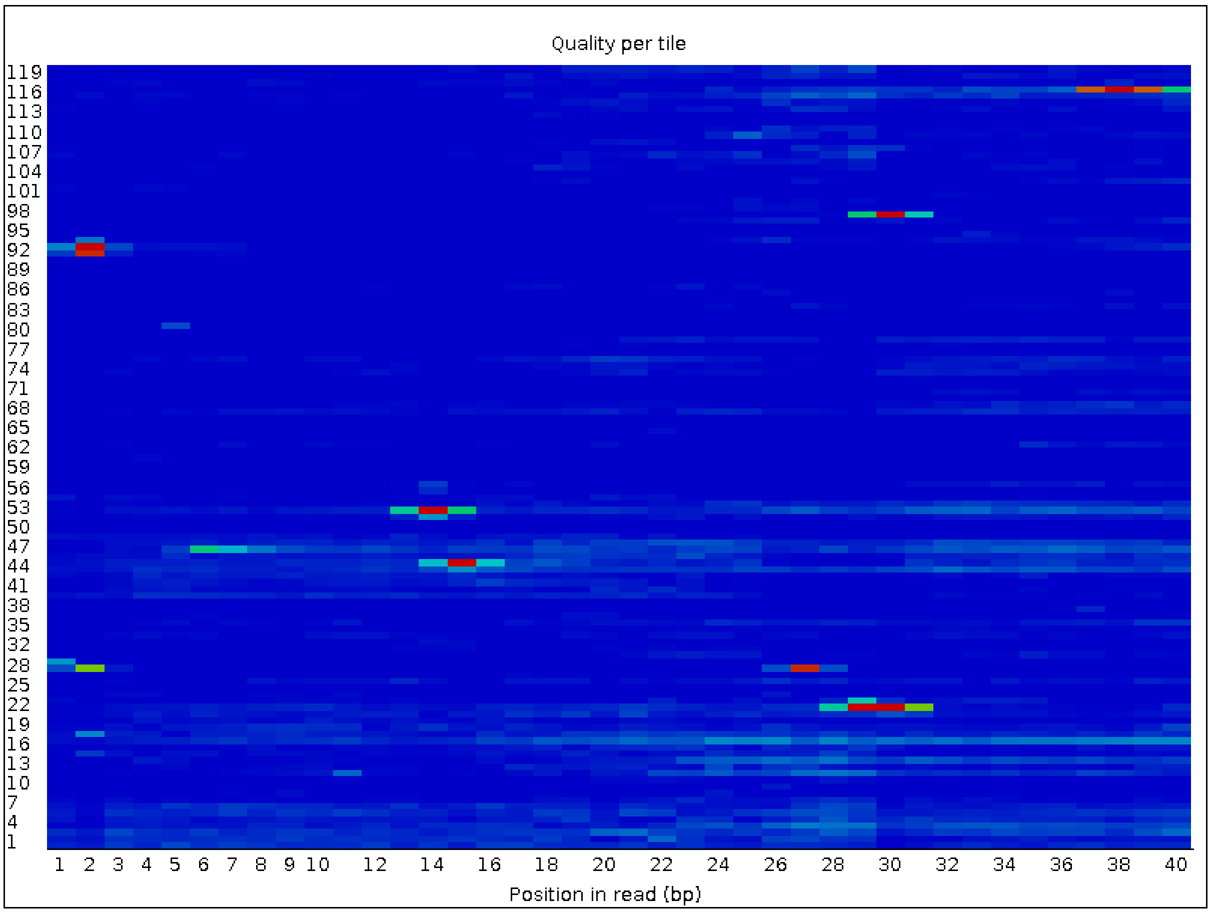


Fig. 3.4: Quality per tile.

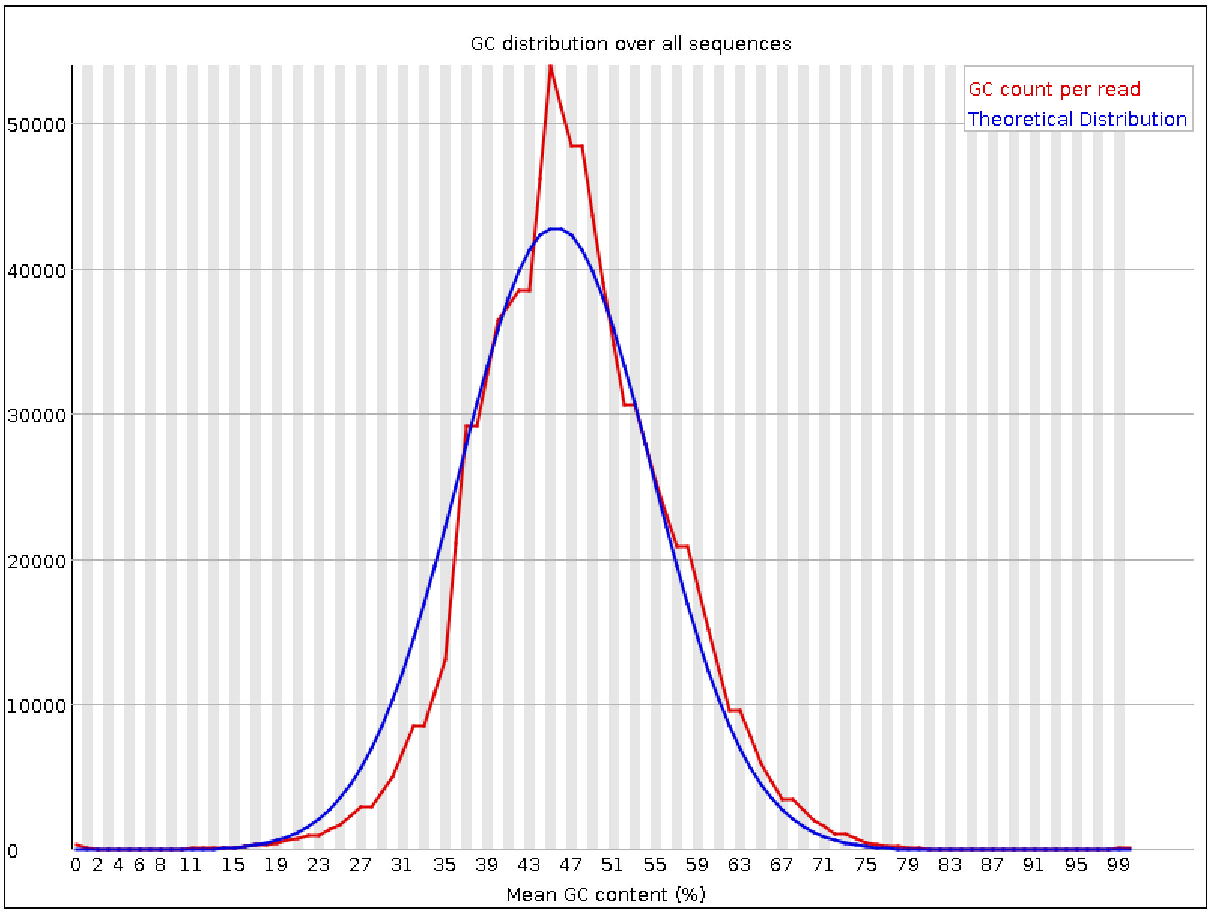


Fig. 3.5: GC distribution over all sequences.

CHAPTER

FOUR

GENOME ASSEMBLY

4.1 Preface

In this section we will use our skill on the command-line interface to create a genome assembly from sequencing data.

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

4.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 4.1.

4.3 Learning outcomes

After studying this tutorial you should be able to:

1. Compute and interpret a whole genome assembly.
2. Judge the quality of a genome assembly.

4.4 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
data/
multiqc_data/
multiqc_report.html
trimmed/
trimmed-fastqc/
```

Attention: If you have not run the previous section *Quality control* (page 9), you can download the trimmed data needed for this section here: *Downloads* (page 77). Download the file to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O trimmed.tar.gz https://osf.io/m3wpr/download
tar xvzf trimmed.tar.gz
```

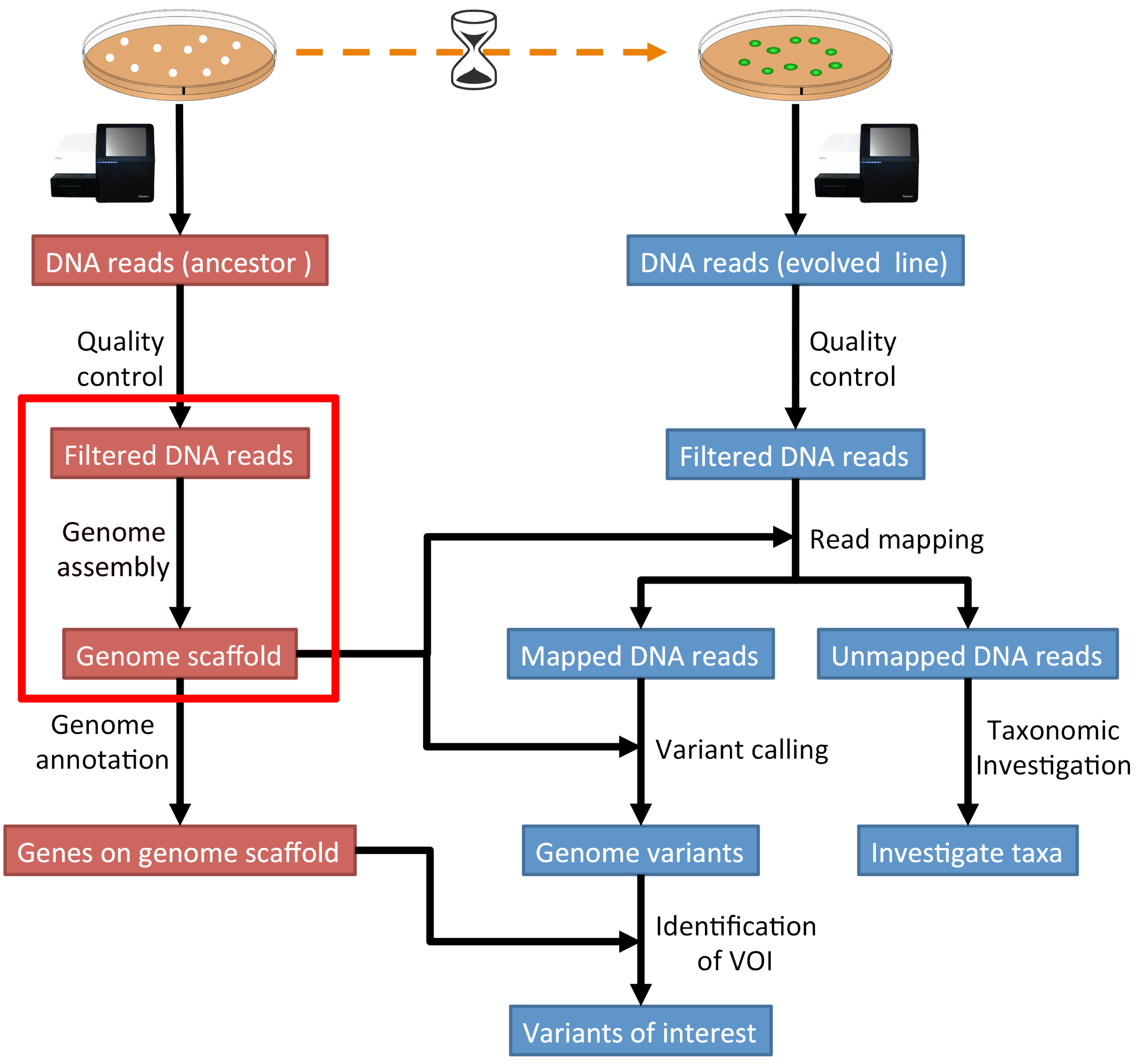


Fig. 4.1: The part of the workflow we will work on in this section marked in red.

4.5 Creating a genome assembly

We want to create a genome assembly for our ancestor. We are going to use the quality trimmed forward and backward DNA sequences and use a program called SPAdes[45] to build a genome assembly.

Todo:

1. Discuss briefly why we are using the ancestral sequences to create a reference genome as opposed to the evolved line.

4.5.1 Installing the software

We are going to use a program called SPAdes[46] fo assembling our genome. In a recent evaluation of assembly software, SPAdes[47] was found to be a good choice for fungal genomes [ABBAS2014]. It is also simple to install and use.

```
$ conda create -n assembly spades quast
$ conda activate assembly
```

4.5.2 SPAdes usage

```
# change to your analysis root folder
$ cd ~/analysis

# first create a output directory for the assemblies
$ mkdir assembly

# to get a help for spades and an overview of the parameter type:
$ spades.py -h
```

Generally, paired-end data is submitted in the following way to SPAdes[48]:

```
$ spades.py -o result-directory -1 read1.fastq.gz -2 read2.fastq.gz
```

Todo:

1. Run SPAdes[49] with default parameters on the ancestor's trimmed reads
2. Read in the SPAdes[50] manual about about assembling with 2x150bp reads
3. Run SPAdes[51] a second time but use the options suggested at the SPAdes[52] manual section 3.4[53] for assembling 2x150bp paired-end reads. Use a different output directory `assembly/spades-150` for this run.

[45] http://bioinf.spbau.ru/spades
[46] http://bioinf.spbau.ru/spades
[47] http://bioinf.spbau.ru/spades
[48] http://bioinf.spbau.ru/spades
[49] http://bioinf.spbau.ru/spades
[50] http://bioinf.spbau.ru/spades
[51] http://bioinf.spbau.ru/spades
[52] http://bioinf.spbau.ru/spades
[53] http://cab.spbu.ru/files/release3.14.0/manual.html#sec3.4

Hint: Should you not get it right, try the commands in *Code: SPAdes assembly (trimmed data)* (page 74).

4.6 Assembly quality assessment

4.6.1 Assembly statistics

Quast[54] (QUality ASsessment Tool) [GUREVICH2013], evaluates genome assemblies by computing various metrics, including:

- N50: length for which the collection of all contigs of that length or longer covers at least 50% of assembly length
- NG50: where length of the reference genome is being covered
- NA50 and NGA50: where aligned blocks instead of contigs are taken
- miss-assemblies: miss-assembled and unaligned contigs or contigs bases
- genes and operons covered

It is easy with Quast[55] to compare these measures among several assemblies. The program can be used on their website[56].

```
$ conda install quast
```

Run Quast[57] with both assembly scaffolds.fasta files to compare the results.

```
$ quast -o assembly/quast assembly/spades-default/scaffolds.fasta assembly/spades-150/scaffolds.
↪fasta
```

Todo:

1. Compare the results of Quast[58] with regards to the two different assemblies.
2. Which one do you prefer and why?

4.7 Compare the untrimmed data

Todo:

1. To see if our trimming procedure has an influence on our assembly, run the same command you used on the trimmed data on the original untrimmed data.
2. Run Quast[59] on the assembly and compare the statistics to the one derived for the trimmed data set. Write down your observations.

[54] http://quast.bioinf.spbau.ru/
[55] http://quast.bioinf.spbau.ru/
[56] http://quast.bioinf.spbau.ru/
[57] http://quast.bioinf.spbau.ru/
[58] http://quast.bioinf.spbau.ru/
[59] http://quast.bioinf.spbau.ru/

Hint: Should you not get it right, try the commands in *Code: SPAdes assembly (original data)* (page 74).

4.8 Further reading

4.8.1 Background on Genome Assemblies

- How to apply de Bruijn graphs to genome assembly. [COMPEAU2011]
- Sequence assembly demystified. [NAGARAJAN2013]

4.8.2 Evaluation of Genome Assembly Software

- GAGE: A critical evaluation of genome assemblies and assembly algorithms. [SALZBERG2012]
- Assessment of de novo assemblers for draft genomes: a case study with fungal genomes. [ABBAS2014]

4.9 Web links

- Lectures for this topic: Genome Assembly: An Introduction[60]
- SPAdes[61]
- Quast[62]
- Bandage[63] (Bioinformatics Application for Navigating De novo Assembly Graphs Easily) is a program that visualizes a genome assembly as a graph [WICK2015].

[60] https://dx.doi.org/10.6084/m9.figshare.2972323.v1
[61] http://bioinf.spbau.ru/spades
[62] http://quast.bioinf.spbau.ru/
[63] https://rrwick.github.io/Bandage/

CHAPTER

FIVE

READ MAPPING

5.1 Preface

In this section we will use our skill on the command-line interface to map our reads from the evolved line to our ancestral reference genome.

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

5.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 5.1.

5.3 Learning outcomes

After studying this section of the tutorial you should be able to:

1. Explain the process of sequence read mapping.
2. Use bioinformatics tools to map sequencing reads to a reference genome.
3. Filter mapped reads based on quality.

5.4 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
# create a mapping result directory
$ mkdir mappings
$ ls -1F
```

```
assembly/
data/
mappings/
multiqc_data/
multiqc_report.html
trimmed/
trimmed-fastqc/
```

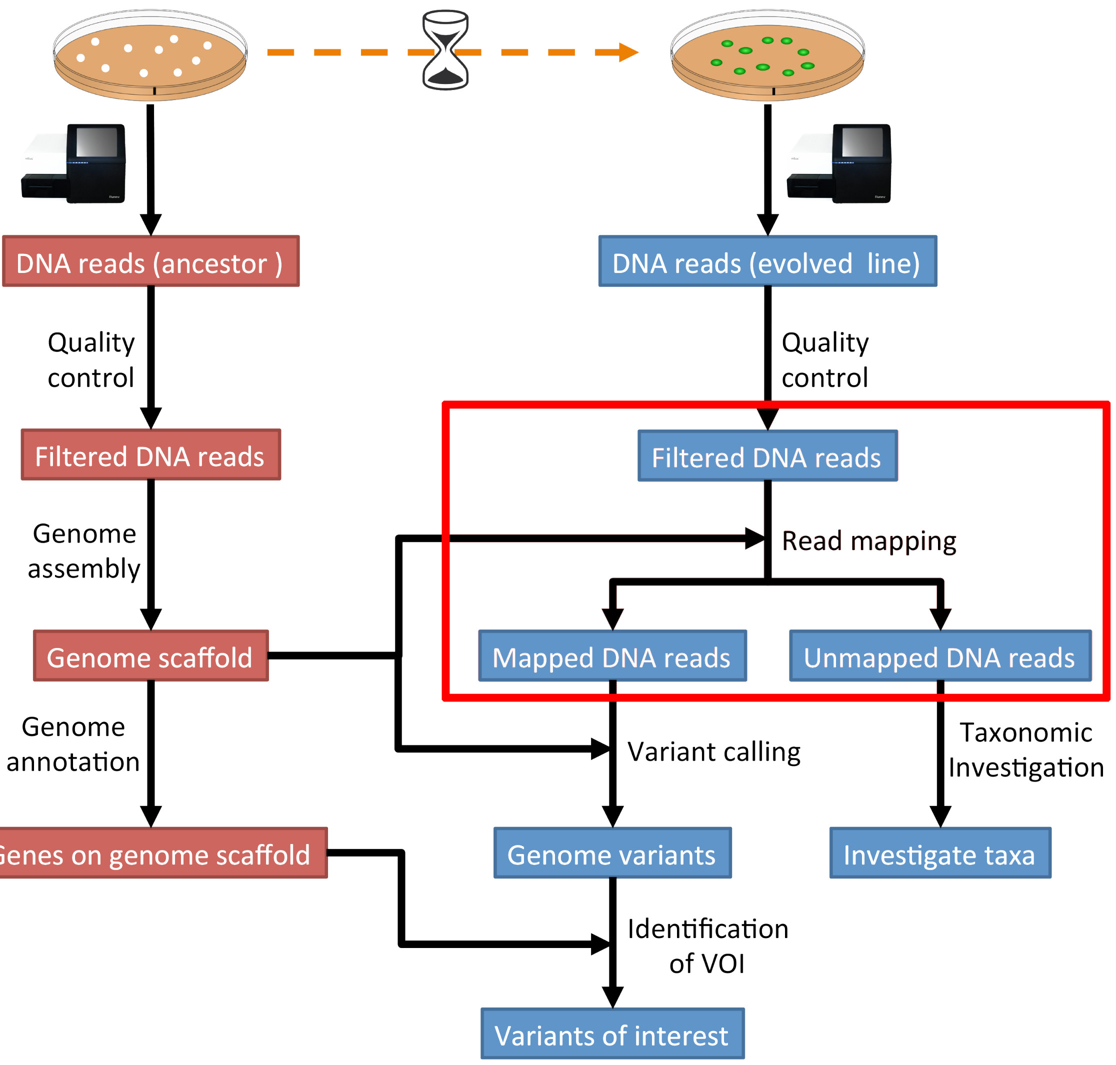


Fig. 5.1: The part of the workflow we will work on in this section marked in red.

Attention: If you have not run the previous sections on *Quality control* (page 9) and *Genome assembly* (page 19), you can download the trimmed data and the genome assembly needed for this section here: *Downloads* (page 77). Download the files to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O trimmed.tar.gz https://osf.io/m3wpr/download
tar xvzf trimmed.tar.gz
wget -O assembly.tar.gz https://osf.io/t2zpm/download
tar xvzf assembly.tar.gz
```

5.5 Mapping sequence reads to a reference genome

We want to map the sequencing reads to the ancestral reference genome. We are going to use the quality trimmed forward and backward DNA sequences of the evolved line and use a program called BWA[70] to map the reads.

Todo:

1. Discuss briefly why we are using the ancestral genome as a reference genome as opposed to a genome for the evolved line.

5.5.1 Downloading the reference genome assembly

Todo: In the assembly section at "*Genome assembly* (page 19)", we created a genome assembly. However, we actually used sub-sampled data as otherwise the assemblies would have taken a long time to finish. To continue, please download the assembly created on the complete dataset (*Downloads* (page 77)). Unarchive and uncompress the files with `tar -xvzf assembly.tar.gz`.

5.5.2 Installing the software

We are going to use a program called BWA[71] to map our reads to our genome.

It is simple to install and use.

```
$ conda create --yes -n mapping samtools bwa qualimap r-base
$ conda activate mapping
```

[70] http://bio-bwa.sourceforge.net/
[71] http://bio-bwa.sourceforge.net/

5.6 BWA

5.6.1 Overview

BWA[72] is a short read aligner, that can take a reference genome and map single- or paired-end sequence data to it [LI2009]. It requires an indexing step in which one supplies the reference genome and BWA[73] will create an index that in the subsequent steps will be used for aligning the reads to the reference genome. While this step can take some time, the good thing is the index can be reused over and over. The general command structure of the BWA[74] tools we are going to use are shown below:

```
# bwa index help
$ bwa index

# indexing
$ bwa index path/to/reference-genome.fa

# bwa mem help
$ bwa mem

# single-end mapping, general command structure, adjust to your case
$ bwa mem path/to/reference-genome.fa path/to/reads.fq.gz > path/to/aln-se.sam

# paired-end mapping, general command structure, adjust to your case
$ bwa mem path/to/reference-genome.fa path/to/read1.fq.gz path/to/read2.fq.gz > path/to/aln-pe.sam
```

5.6.2 Creating a reference index for mapping

Todo: Create an BWA[75] index for our reference genome assembly. Attention! Remember which file you need to submit to BWA[76].

Hint: Should you not get it right, try the commands in *Code: BWA indexing* (page 74).

Note: Should you be unable to run BWA[77] indexing on the data, you can download the index from *Downloads* (page 77). Unarchive and uncompress the files with `tar -xvzf bwa-index.tar.gz`.

5.6.3 Mapping reads in a paired-end manner

Now that we have created our index, it is time to map the trimmed sequencing reads of our two evolved line to the reference genome.

Todo: Use the correct `bwa mem` command structure from above and map the reads of the two evolved line to the reference genome.

[72] http://bio-bwa.sourceforge.net/
[73] http://bio-bwa.sourceforge.net/
[74] http://bio-bwa.sourceforge.net/
[75] http://bio-bwa.sourceforge.net/
[76] http://bio-bwa.sourceforge.net/
[77] http://bio-bwa.sourceforge.net/

Hint: Should you not get it right, try the commands in *Code: BWA mapping* (page 74).

5.7 The sam mapping file-format

BWA[78], like most mappers, will produce a mapping file in sam-format. Have a look into the sam-file that was created by either program. A quick overview of the sam-format can be found here[79] and even more information can be found here[80]. Briefly, first there are a lot of header lines. Then, for each read, that mapped to the reference, there is one line.

The columns of such a line in the mapping file are described in Table 5.1.

Table 5.1: The sam-file format fields.

Col	Field	Description
1	QNAME	Query (pair) NAME
2	FLAG	bitwise FLAG
3	RNAME	Reference sequence NAME
4	POS	1-based leftmost POSition/coordinate of clipped sequence
5	MAPQ	MAPping Quality (Phred-scaled)
6	CIAGR	extended CIGAR string
7	MRNM	Mate Reference sequence NaMe ('=' if same as RNAME)
8	MPOS	1-based Mate POSition
9	ISIZE	Inferred insert SIZE
10	SEQ	query SEQuence on the same strand as the reference
11	QUAL	query QUALity (ASCII-33 gives the Phred base quality)
12	OPT	variable OPTional fields in the format TAG:VTYPE:VALUE

One line of a mapped read can be seen here:

```
M02810:197:000000000-AV55U:1:1101:10000:11540   83      NODE_1_length_1419525_cov_15.3898      ␣
↪607378  60      151M    =       607100  -429   ␣
↪TATGGTATCACTTATGGTATCACTTATGGCTATCACTAATGGCTATCACTTATGGTATCACTTATGACTATCAGACGTTATTACTATCAGACGATAACTATCAGACTTTATTACT
↪FHGHHHHHGGGHHHHHHHHHHHHHHHHHHHHHGHHHHHHHHHHHHGHHHHHGHHHHHHHHHGDHHHHHHHHHGHHHHGHHHGHHHHHHHFHHHHGHHHHIHHHHHHHHHHHHHHHHHHHHHHGHH
↪NM:i:0  MD:Z:151        AS:i:151        XS:i:0
```

It basically defines the read and the position within the reference genome, where the read mapped and a quality of the mapping.

5.8 Mapping post-processing

5.8.1 Fix mates and compress

Because aligners can sometimes leave unusual SAM flag[81] information on SAM records, it is helpful when working with many tools to first clean up read pairing information and flags with SAMtools[82]. We are going to produce also compressed bam output for efficient storing of and access to the mapped reads. Note, `samtools fixmate` expects **name-sorted** input files, which we can achieve with `samtools sort -n`.

```
$ samtools sort -n -O sam mappings/evol1.sam | samtools fixmate -m -O bam - mappings/evol1.fixmate.
↪bam
```

[78] http://bio-bwa.sourceforge.net/
[79] http://bio-bwa.sourceforge.net/bwa.shtml#4
[80] http://samtools.github.io/hts-specs/SAMv1.pdf
[81] http://bio-bwa.sourceforge.net/bwa.shtml#4
[82] http://samtools.sourceforge.net/

- `-m`: Add ms (mate score) tags. These are used by markdup (below) to select the best reads to keep.
- `-O bam`: specifies that we want compressed bam output from fixmate

Attention: The step of sam to bam-file conversion might take a few minutes to finish, depending on how big your mapping file is.

We will be using the SAM flag[83] information later below to extract specific alignments.

Hint: A very useful tools to explain flags can be found here[84].

Once we have `bam`-file, we can also delete the original `sam`-file as it requires too much space and we can always recreate it from the `bam`-file.

```
$ rm mappings/evol1.sam
```

5.8.2 Sorting

We are going to use SAMtools[85] again to sort the bam-file into **coordinate order**:

```
# convert to bam file and sort
$ samtools sort -O bam -o mappings/evol1.sorted.bam mappings/evol1.fixmate.bam

# Once it successfully finished, delete the fixmate file to save space
$ rm mappings/evol1.fixmate.bam
```

- `-o`: specifies the name of the output file.
- `-O bam`: specifies that the output will be bam-format

5.8.3 Remove duplicates

In this step we remove duplicate reads. The main purpose of removing duplicates is to mitigate the effects of PCR amplification bias introduced during library construction. **It should be noted that this step is not always recommended.** It depends on the research question. In SNP calling it is a good idea to remove duplicates, as the statistics used in the tools that call SNPs sub-sequently expect this (most tools anyways). However, for other research questions that use mapping, you might not want to remove duplicates, e.g. RNA-seq.

```
$ samtools markdup -r -S mappings/evol1.sorted.bam mappings/evol1.sorted.dedup.bam

# if it worked, delete the original file
$ rm mappings/evol1.sorted.bam
```

Todo: Figure out what "PCR amplification bias" means.

Note: Should you be unable to do the post-processing steps, you can download the mapped data from *Downloads* (page 77).

[83] http://bio-bwa.sourceforge.net/bwa.shtml#4
[84] http://broadinstitute.github.io/picard/explain-flags.html
[85] http://samtools.sourceforge.net/

5.9 Mapping statistics

5.9.1 Stats with SAMtools

Lets get an mapping overview:

```
$ samtools flagstat mappings/evol1.sorted.dedup.bam
```

Todo: Look at the mapping statistics and understand their meaning[86]. Discuss your results. Explain why we may find mapped reads that have their mate mapped to a different chromosome/contig? Can they be used for something?

For the sorted bam-file we can get read depth for at all positions of the reference genome, e.g. how many reads are overlapping the genomic position.

```
$ samtools depth mappings/evol1.sorted.dedup.bam | gzip > mappings/evol1.depth.txt.gz
```

Todo: Extract the depth values for contig 20 and load the data into R, calculate some statistics of our scaffold.

```
$ zcat mappings/evol1.depth.txt.gz | egrep '^NODE_20_' | gzip >  mappings/NODE_20.depth.txt.gz
```

Now we quickly use some R[87] to make a coverage plot for contig NODE20. Open a R[88] shell by typing `R` on the command-line of the shell.

```
x <- read.table('mappings/NODE_20.depth.txt.gz', sep='\t', header=FALSE,  strip.white=TRUE)

# Look at the beginning of x
head(x)

# calculate average depth
mean(x[,3])
# std dev
sqrt(var(x[,3]))

# mark areas that have a coverage below 20 in red
plot(x[,2], x[,3], col = ifelse(x[,3] < 20,'red','black'), pch=19, xlab='postion', ylab='coverage')

# to save a plot
png('mappings/covNODE20.png', width = 1200, height = 500)
plot(x[,2], x[,3], col = ifelse(x[,3] < 20,'red','black'), pch=19, xlab='postion', ylab='coverage')
dev.off()
```

The result plot will be looking similar to the one in Fig. 5.2

Todo: Look at the created plot. Explain why it makes sense that you find relatively bad coverage at the beginning and the end of the contig.

[86] https://www.biostars.org/p/12475/
[87] https://www.r-project.org/
[88] https://www.r-project.org/

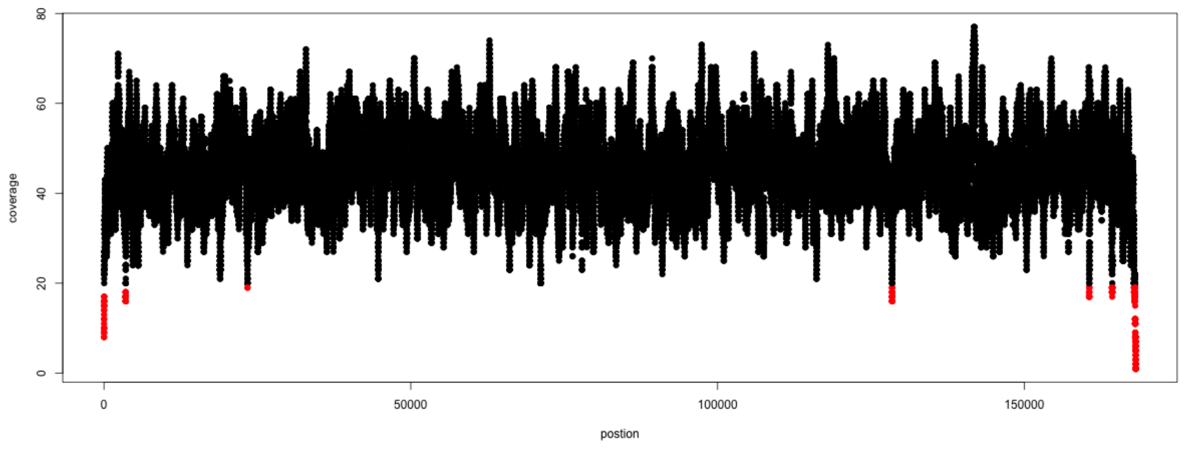


Fig. 5.2: A example coverage plot for a contig with highlighted in red regions with a coverage below 20 reads.

5.9.2 Stats with QualiMap

For a more in depth analysis of the mappings, one can use QualiMap[89] [OKO2015].

QualiMap[90] examines sequencing alignment data in SAM/BAM files according to the features of the mapped reads and provides an overall view of the data that helps to the detect biases in the sequencing and/or mapping of the data and eases decision-making for further analysis.

Run QualiMap[91] with:

```
$ qualimap bamqc -bam mappings/evol1.sorted.dedup.bam
# Once finsished open reult page with
$ firefox mappings/evol1.sorted.dedup_stats/qualimapReport.html
```

This will create a report in the mapping folder. See this webpage[92] to get help on the sections in the report.

Todo: Investigate the mapping of the evolved sample. Write down your observations.

5.10 Sub-selecting reads

It is important to remember that the mapping commands we used above, without additional parameters to sub-select specific alignments (e.g. for Bowtie2[93] there are options like `--no-mixed`, which suppresses unpaired alignments for paired reads or `--no-discordant`, which suppresses discordant alignments for paired reads, etc.), are going to output all reads, including unmapped reads, multi-mapping reads, unpaired reads, discordant read pairs, etc. in one file. We can sub-select from the output reads we want to analyse further using SAMtools[94].

[89] http://qualimap.bioinfo.cipf.es/
[90] http://qualimap.bioinfo.cipf.es/
[91] http://qualimap.bioinfo.cipf.es/
[92] http://qualimap.bioinfo.cipf.es/doc_html/analysis.html#output
[93] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[94] http://samtools.sourceforge.net/

Todo: Explain what concordant and discordant read pairs are? Look at the Bowtie2[95] manual.

5.10.1 Concordant reads

We can select read-pair that have been mapped in a correct manner (same chromosome/contig, correct orientation to each other, distance between reads is not stupid).

Attention: We show the command here, but we are not going to use it.

```
$ samtools view -h -b -f 3 mappings/evol1.sorted.dedup.bam > mappings/evol1.sorted.dedup.concordant.
↪bam
```

- `-b`: Output will be bam-format
- `-f 3`: Only extract correctly paired reads. `-f` extracts alignments with the specified SAM flag[96] set.

Todo: Our final aim is to identify variants. For a particular class of variants, it is not the best idea to only focus on concordant reads. Why is that?

5.10.2 Quality-based sub-selection

In this section we want to sub-select reads based on the quality of the mapping. It seems a reasonable idea to only keep good mapping reads. As the SAM-format contains at column 5 the $MAPQ$ value, which we established earlier is the "MAPping Quality" in Phred-scaled, this seems easily achieved. The formula to calculate the $MAPQ$ value is: $MAPQ = -10 * log10(p)$, where p is the probability that the read is mapped wrongly. However, there is a problem! **While the MAPQ information would be very helpful indeed, the way that various tools implement this value differs.** A good overview can be found here[97]. Bottom-line is that we need to be aware that different tools use this value in different ways and the it is good to know the information that is encoded in the value. Once you dig deeper into the mechanics of the $MAPQ$ implementation it becomes clear that this is not an easy topic. If you want to know more about the $MAPQ$ topic, please follow the link above.

For the sake of going forward, we will sub-select reads with at least medium quality as defined by Bowtie2[98]:

```
$ samtools view -h -b -q 20 mappings/evol1.sorted.dedup.bam > mappings/evol1.sorted.dedup.q20.bam
```

- `-h`: Include the sam header
- `-q 20`: Only extract reads with mapping quality $>=$ 20

Hint: I will repeat here a recommendation given at the source link[99] above, as it is a good one: If you unsure what $MAPQ$ scoring scheme is being used in your own data then you can plot out the $MAPQ$ distribution in a BAM file using programs like the mentioned QualiMap[100] or similar programs. This will at least show you the range and frequency with which different $MAPQ$ values appear and may help identify a suitable threshold you may want to use.

[95] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[96] http://bio-bwa.sourceforge.net/bwa.shtml#4
[97] https://sequencing.qcfail.com/articles/mapq-values-are-really-useful-but-their-implementation-is-a-mess/
[98] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[99] https://sequencing.qcfail.com/articles/mapq-values-are-really-useful-but-their-implementation-is-a-mess/
[100] http://qualimap.bioinfo.cipf.es/

Todo: Please repeat the whole process for the second evolved strain => mapping and post-processing.

Note: Should you be unable to process the second evolved strain look at the coding solutions here: *Code: Mapping post-processing* (page 74)

5.10.3 Unmapped reads

We could decide to use Kraken2[101] like in section *Taxonomic investigation* (page 35) to classify all unmapped sequence reads and identify the species they are coming from and test for contamination.

Lets see how we can get the unmapped portion of the reads from the bam-file:

```
$ samtools view -b -f 4 mappings/evol1.sorted.dedup.bam > mappings/evol1.sorted.unmapped.bam
# we are deleting the original to save space,
# however, in reality you might want to save it to investigate later
$ rm mappings/evol1.sorted.dedup.bam

# count the unmapped reads
$ samtools view -c mappings/evol1.sorted.unmapped.bam
```

- `-b`: indicates that the output is BAM.
- `-f INT`: only include reads with this SAM flag[102] set. You can also use the command `samtools flags` to get an overview of the flags.
- `-c`: count the reads

Lets extract the fastq sequence of the unmapped reads for read1 and read2.

```
$ samtools fastq -1 mappings/evol1.sorted.unmapped.R1.fastq.gz -2 mappings/evol1.sorted.unmapped.R2.
↪fastq.gz mappings/evol1.sorted.unmapped.bam
# delete not needed files
$ rm mappings/evol1.sorted.unmapped.bam
```

[101] https://www.ccb.jhu.edu/software/kraken2/
[102] http://bio-bwa.sourceforge.net/bwa.shtml#4

CHAPTER

SIX

TAXONOMIC INVESTIGATION

6.1 Preface

We want to investigate if there are sequences of other species in our collection of sequenced DNA pieces. We hope that most of them are from our species that we try to study, i.e. the DNA that we have extracted and amplified. This might be a way of quality control, e.g. have the samples been contaminated? Lets investigate if we find sequences from other species in our sequence set.

We will use the tool Kraken2[105] to assign taxonomic classifications to our sequence reads. Let us see if we can id some sequences from other species.

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

6.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 6.1.

6.3 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
assembly/
data/
mappings/
multiqc_data
trimmed/
trimmed-fastqc/
```

Attention: If you have not run the previous section *Read mapping* (page 25), you can download the unmapped sequencing data needed for this section here: *Downloads* (page 77). Download the file to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O mappings.tar.gz https://osf.io/g5at8/download
tar xvzf mappings.tar.gz
```

[105] https://www.ccb.jhu.edu/software/kraken2/

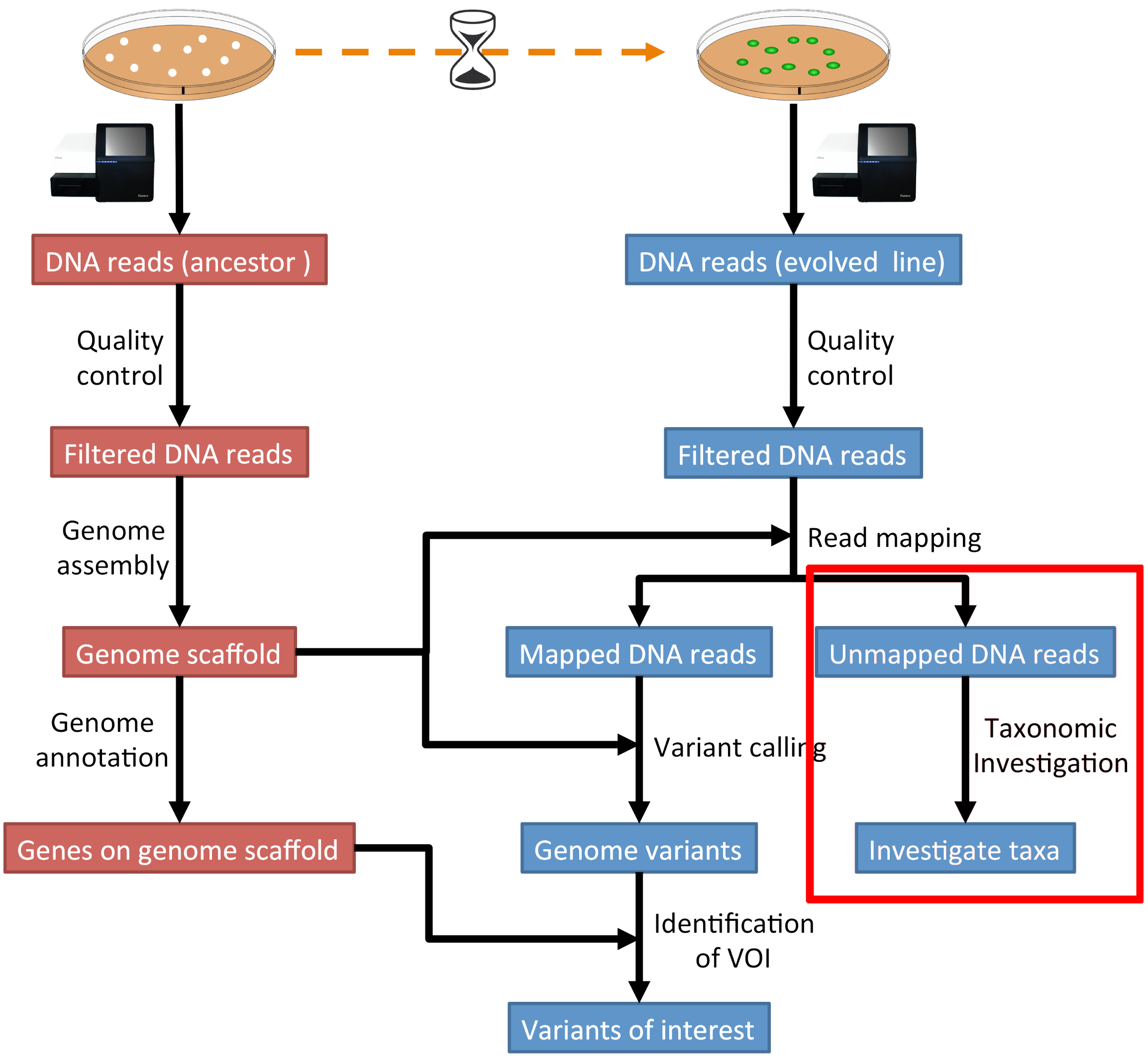


Fig. 6.1: The part of the workflow we will work on in this section marked in red.

6.4 Kraken2

We will be using a tool called Kraken2[106] [WOOD2014]. This tool uses k-mers to assign a taxonomic labels in form of NCBI Taxonomy[107] to the sequence (if possible). The taxonomic label is assigned based on similar k-mer content of the sequence in question to the k-mer content of reference genome sequence. The result is a classification of the sequence in question to the most likely taxonomic label. If the k-mer content is not similar to any genomic sequence in the database used, it will not assign any taxonomic label.

6.4.1 Installation

Use conda in the same fashion as before to install Kraken2[108]. However, we are going to install kraken into its own environment:

```
$ conda create --yes -n kraken kraken2 bracken
$ conda activate kraken
```

Now we create a directory where we are going to do the analysis and we will change into that directory too.

```
# make sure you are in your analysis root folder
$ cd ~/analysis

# create dir
$ mkdir kraken
$ cd kraken
```

Now we need to create or download a Kraken2[109] database that can be used to assign the taxonomic labels to sequences. We opt for downloading the pre-build "minikraken2" database from the Kraken2[110] website:

```
$ curl -O ftp://ftp.ccb.jhu.edu/pub/data/kraken2_dbs/minikraken2_v2_8GB_201904_UPDATE.tgz

# alternatively we can use wget
$ wget ftp://ftp.ccb.jhu.edu/pub/data/kraken2_dbs/minikraken2_v2_8GB_201904_UPDATE.tgz

# once the download is finished, we need to extract the archive content:
$ tar -xvzf minikraken2_v2_8GB_201904_UPDATE.tgz
```

Attention: Should the download fail. Please find links to alternative locations on the *Downloads* (page 77) page.

Note: The "minikraken2" database was created from bacteria, viral and archaea sequences. What are the implications for us when we are trying to classify our sequences?

[106] https://www.ccb.jhu.edu/software/kraken2/
[107] https://www.ncbi.nlm.nih.gov/taxonomy
[108] https://www.ccb.jhu.edu/software/kraken2/
[109] https://www.ccb.jhu.edu/software/kraken2/
[110] https://www.ccb.jhu.edu/software/kraken2/

6.4.2 Usage

Now that we have installed Kraken2[111] and downloaded and extracted the minikraken2 database, we can attempt to investigate the sequences we got back from the sequencing provider for other species as the one it should contain. We call the Kraken2[112] tool and specify the database and fasta-file with the sequences it should use. The general command structure looks like this:

```
$ kraken2 --use-names --threads 4 --db PATH_TO_DB_DIR --report example.report.txt example.fa >␣
↪example.kraken
```

However, we may have fastq-files, so we need to use `--fastq-input` which tells Kraken2[113] that it is dealing with fastq-formated files. The `--gzip-compressed` flag specifies tat te input-files are compressed. In addition, we are dealing with paired-end data, which we can tell Kraken2[114] with the switch `--paired`. Here, we are investigating one of the unmapped paired-end read files of the evolved line.

```
$ kraken2 --use-names --threads 4 --db minikraken2_v2_8GB_201904_UPDATE --fastq-input --report␣
↪evol1 --gzip-compressed --paired ../mappings/evol1.sorted.unmapped.R1.fastq.gz ../mappings/evol1.
↪sorted.unmapped.R2.fastq.gz > evol1.kraken
```

This classification may take a while, depending on how many sequences we are going to classify. The resulting content of the file `evol1.kraken` looks similar to the following example:

```
C       7001326F:121:CBVVLANXX:1:1105:2240:12640        816     251     816:9 171549:5 816:5␣
↪171549:3 2:2 816:5 171549:4 816:34 171549:8 816:4 171549:2 816:10 A:35 816:10 171549:2 816:4␣
↪171549:8 816:34 171549:4 816:5 2:2 171549:3 816:5 171549:5 816:9
C       7001326F:121:CBVVLANXX:1:1105:3487:12536        1339337 202     1339337:67 A:35 1339337:66
U       7001326F:121:CBVVLANXX:1:1105:5188:12504        0       251     0:91 A:35 0:91
U       7001326F:121:CBVVLANXX:1:1105:11030:12689       0       251     0:91 A:35 0:91
U       7001326F:121:CBVVLANXX:1:1105:7157:12806        0       206     0:69 A:35 0:68
```

Each sequence classified by Kraken2[115] results in a single line of output. Output lines contain five tab-delimited fields; from left to right, they are:

1. `C`/`U`: one letter code indicating that the sequence was either classified or unclassified.
2. The sequence ID, obtained from the FASTA/FASTQ header.
3. The taxonomy ID Kraken2[116] used to label the sequence; this is **0** if the sequence is unclassified and otherwise should be the NCBI Taxonomy[117] identifier.
4. The length of the sequence in bp.
5. A space-delimited list indicating the lowest common ancestor (in the taxonomic tree) mapping of each k-mer in the sequence. For example, `562:13 561:4 A:31 0:1 562:3` would indicate that:
    - the first 13 k-mers mapped to taxonomy ID #562
    - the next 4 k-mers mapped to taxonomy ID #561
    - the next 31 k-mers contained an ambiguous nucleotide
    - the next k-mer was not in the database
    - the last 3 k-mers mapped to taxonomy ID #562

[111] https://www.ccb.jhu.edu/software/kraken2/
[112] https://www.ccb.jhu.edu/software/kraken2/
[113] https://www.ccb.jhu.edu/software/kraken2/
[114] https://www.ccb.jhu.edu/software/kraken2/
[115] https://www.ccb.jhu.edu/software/kraken2/
[116] https://www.ccb.jhu.edu/software/kraken2/
[117] https://www.ncbi.nlm.nih.gov/taxonomy

Note: The Kraken2[118] manual can be accessed here[119].

6.4.3 Investigate taxa

We can use the webpage NCBI TaxIdentifier[120] to quickly get the names to the taxonomy identifier. However, this is impractical as we are dealing potentially with many sequences. Kraken2[121] has some scripts that help us understand our results better.

Because we used the Kraken2[122] switch `--report FILE`, we have got also a sample-wide report of all taxa found. This is much better to get an overview what was found.

The first few lines of an example report are shown below.

```
83.56  514312  514312  U       0       unclassified
16.44  101180  0       R       1       root
16.44  101180  0       R1      131567    cellular organisms
16.44  101180  2775    D       2           Bacteria
13.99  86114   1       D1      1783270       FCB group
13.99  86112   0       D2      68336           Bacteroidetes/Chlorobi group
13.99  86103   8       P       976               Bacteroidetes
13.94  85798   2       C       200643              Bacteroidia
13.94  85789   19      O       171549                Bacteroidales
13.87  85392   0       F       815                     Bacteroidaceae
```

The output of kraken-report is tab-delimited, with one line per taxon. The fields of the output, from left-to-right, are as follows:

1. **Percentage** of reads covered by the clade rooted at this taxon
2. **Number of reads** covered by the clade rooted at this taxon
3. **Number of reads** assigned directly to this taxon
4. A rank code, indicating **(U)nclassified, (D)omain, (K)ingdom, (P)hylum, (C)lass, (O)rder, (F)amily, (G)enus, or (S)pecies**. All other ranks are simply **"-"**.
5. NCBI Taxonomy[123] ID
6. The indented scientific name

Note: If you want to compare the taxa content of different samples to another, one can create a report whose structure is always the same for all samples, disregarding which taxa are found (obviously the percentages and numbers will be different).

We can cerate such a report using the option `--report-zero-counts` which will print out all taxa (instead of only those found). We then sort the taxa according to taxa-ids (column 5), e.g. `sort -n -k5`.

The report is not ordered according to taxa ids and contains all taxa in the database, even if they have not been found in our sample and are thus zero. The columns are the same as in the former report, however, we have more rows and they are now differently sorted, according to the NCBI Taxonomy[124] id.

[118] https://www.ccb.jhu.edu/software/kraken2/
[119] https://www.ccb.jhu.edu/software/kraken2/index.shtml?t=manual
[120] https://www.ncbi.nlm.nih.gov/Taxonomy/TaxIdentifier/tax_identifier.cgi
[121] https://www.ccb.jhu.edu/software/kraken2/
[122] https://www.ccb.jhu.edu/software/kraken2/
[123] https://www.ncbi.nlm.nih.gov/taxonomy
[124] https://www.ncbi.nlm.nih.gov/taxonomy

6.4.4 Bracken

Bracken[125] stands for Bayesian Re-estimation of Abundance with KrakEN, and is a statistical method that computes the abundance of species in DNA sequences from a metagenomics sample [LU2017]. Bracken[126] uses the taxonomy labels assigned by Kraken2[127] (see above) to estimate the number of reads originating from each species present in a sample. Bracken[128] classifies reads to the best matching location in the taxonomic tree, but does not estimate abundances of species. Combined with the Kraken classifier, Bracken[129] will produces more accurate species- and genus-level abundance estimates than Kraken2[130] alone.

The use of Bracken[131] subsequent to Kraken2[132] is optional but might improve on the Kraken2[133] results.

Installation

We installed Bracken[134] already together with Kraken2[135] above, so it should be ready to be used. We also downloaded the Bracken[136] files together with the minikraken2 database above, so we are good to go.

Usage

Now, we can use Bracken[137] on the Kraken2[138] results to improve them.

The general structure of the Bracken[139] command look like this:

```
$ bracken -d PATH_TO_DB_DIR -i kraken2.report -o bracken.species.txt -l S
```

- `-l S`: denotes the level we want to look at. `S` stands for species but other levels are available.
- `-d PATH_TO_DB_DIR`: specifies the path to the Kraken2[140] database that should be used.

Let us apply Bracken[141] to the example above:

```
$ bracken -d minikraken2_v2_8GB_201904_UPDATE -i evol1.kraken -l S -o evol1.bracken
```

The species-focused result-table looks similar to this:

```
name    taxonomy_id     taxonomy_lvl    kraken_assigned_reads   added_reads     new_est_reads  ␣
↪fraction_total_reads
Streptococcus sp. oral taxon 431        712633  S       2       0       2       0.00001
Neorhizobium sp. NCHU2750       1825976 S       0       0       0       0.00000
Pseudomonas sp. MT-1    150396  S       0       0       0       0.00000
Ahniella affigens       2021234 S       1       0       1       0.00000
Sinorhizobium sp. CCBAU 05631   794846  S       0       0       0       0.00000
Cohnella sp. 18JY8-7    2480923 S       1       0       1       0.00000
```

(continues on next page)

[125] https://ccb.jhu.edu/software/bracken/index.shtml
[126] https://ccb.jhu.edu/software/bracken/index.shtml
[127] https://www.ccb.jhu.edu/software/kraken2/
[128] https://ccb.jhu.edu/software/bracken/index.shtml
[129] https://ccb.jhu.edu/software/bracken/index.shtml
[130] https://www.ccb.jhu.edu/software/kraken2/
[131] https://ccb.jhu.edu/software/bracken/index.shtml
[132] https://www.ccb.jhu.edu/software/kraken2/
[133] https://www.ccb.jhu.edu/software/kraken2/
[134] https://ccb.jhu.edu/software/bracken/index.shtml
[135] https://www.ccb.jhu.edu/software/kraken2/
[136] https://ccb.jhu.edu/software/bracken/index.shtml
[137] https://ccb.jhu.edu/software/bracken/index.shtml
[138] https://www.ccb.jhu.edu/software/kraken2/
[139] https://ccb.jhu.edu/software/bracken/index.shtml
[140] https://www.ccb.jhu.edu/software/kraken2/
[141] https://ccb.jhu.edu/software/bracken/index.shtml

(continued from previous page)

```
Bacillus velezensis     492670  S       4       4       8       0.00002
Actinoplanes missouriensis      1866    S       2       8       10      0.00002
```

The important column is the new_est_reads, which gives the newly estimated reads.

6.5 Centrifuge

We can also use another tool by the same group called Centrifuge[142] [KIM2017]. This tool uses a novel indexing scheme based on the Burrows-Wheeler transform (BWT) and the Ferragina-Manzini (FM) index, optimized specifically for the metagenomic classification problem to assign a taxonomic labels in form of NCBI Taxonomy[143] to the sequence (if possible). The result is a classification of the sequence in question to the most likely taxonomic label. If the search sequence is not similar to any genomic sequence in the database used, it will not assign any taxonomic label.

Note: I would normally use Kraken2[144] and only prefer Centrifuge[145] if memory and/or speed are an issue .

6.5.1 Installation

Use conda in the same fashion as before to install Centrifuge[146]:

```
$ conda create --yes -n centrifuge centrifuge
$ conda activate centrifuge
```

Now we create a directory where we are going to do the analysis and we will change into that directory too.

```
# make sure you are in your analysis root folder
$ cd ~/analysis

# create dir
$ mkdir centrifuge
$ cd centrifuge
```

Now we need to create or download a Centrifuge[147] database that can be used to assign the taxonomic labels to sequences. We opt for downloading the pre-build database from the Centrifuge[148] website:

```
$ curl -O ftp://ftp.ccb.jhu.edu/pub/infphilo/centrifuge/data/p_compressed+h+v.tar.gz

$ # alternatively we can use wget
$ wget ftp://ftp.ccb.jhu.edu/pub/infphilo/centrifuge/data/p_compressed+h+v.tar.gz

# once the download is finished, we need to extract the archive content
# It will extract a few files from the archive and may take a moment to finish.
$ tar -xvzf p_compressed+h+v.tar.gz
```

142 http://www.ccb.jhu.edu/software/centrifuge/index.shtml
143 https://www.ncbi.nlm.nih.gov/taxonomy
144 https://www.ccb.jhu.edu/software/kraken2/
145 http://www.ccb.jhu.edu/software/centrifuge/index.shtml
146 http://www.ccb.jhu.edu/software/centrifuge/index.shtml
147 http://www.ccb.jhu.edu/software/centrifuge/index.shtml
148 http://www.ccb.jhu.edu/software/centrifuge/index.shtml

Attention: Should the download fail. Please find links to alternative locations on the *Downloads* (page 77) page.

Note: The database we will be using was created from bacteria and archaea sequences only. What are the implications for us when we are trying to classify our sequences?

6.5.2 Usage

Now that we have installed Centrifuge[149] and downloaded and extracted the pre-build database, we can attempt to investigate the sequences we got back from the sequencing provider for other species as the one it should contain. We call the Centrifuge[150] tool and specify the database and fastq-files with the sequences it should use. The general command structure looks like this:

```
$ centrifuge -x p_compressed+h+v -1 example.1.fq -2 example.2.fq -U single.fq --report-file report.
↪txt -S results.txt
```

Here, we are investigating paired-end read files of the evolved line.

```
$ centrifuge -x p_compressed+h+v -1 ../mappings/evol1.sorted.unmapped.R1.fastq  -2 ../mappings/
↪evol1.sorted.unmapped.R2.fastq --report-file evol1-report.txt -S evol1-results.txt
```

This classification may take a moment, depending on how many sequences we are going to classify. The resulting content of the file `evol1-results.txt` looks similar to the following example:

```
readID   seqID    taxID    score    2ndBestScore    hitLength       queryLength     numMatches
M02810:197:000000000-AV55U:1:1101:15316:8461    cid|1747        1747    1892    0       ␣
↪103     135      1
M02810:197:000000000-AV55U:1:1101:15563:3249    cid|161879      161879  18496   0       ␣
↪151     151      1
M02810:197:000000000-AV55U:1:1101:19743:5166    cid|564 564     10404   10404   117     ␣
↪151     2
M02810:197:000000000-AV55U:1:1101:19743:5166    cid|562 562     10404   10404   117     ␣
↪151     2
```

Each sequence classified by Centrifuge[151] results in a single line of output. Output lines contain eight tab-delimited fields; from left to right, they are according to the Centrifuge[152] website:

1. The read ID from a raw sequencing read.
2. The sequence ID of the genomic sequence, where the read is classified.
3. The taxonomic ID of the genomic sequence in the second column.
4. The score for the classification, which is the weighted sum of hits.
5. The score for the next best classification.
6. A pair of two numbers: (1) an approximate number of base pairs of the read that match the genomic sequence and (2) the length of a read or the combined length of mate pairs.
7. A pair of two numbers: (1) an approximate number of base pairs of the read that match the genomic sequence and (2) the length of a read or the combined length of mate pairs.
8. The number of classifications for this read, indicating how many assignments were made.

[149] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[150] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[151] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[152] http://www.ccb.jhu.edu/software/centrifuge/index.shtml

6.5.3 Investigate taxa

Centrifuge report

The command above creates a Centrifuge[153] report automatically for us. It contains an overview of the identified taxa and their abundances in your supplied sequences (normalised to genomic length):

```
name    taxID   taxRank genomeSize      numReads        numUniqueReads  abundance
Pseudomonas aeruginosa  287     species 22457305        1       0       0.0
Pseudomonas fluorescens 294     species 14826544        1       1       0.0
Pseudomonas putida      303     species 6888188 1       1       0.0
Ralstonia pickettii     329     species 6378979 3       2       0.0
Pseudomonas pseudoalcaligenes   330     species 4691662 1       1       0.0171143
```

Each line contains seven tab-delimited fields; from left to right, they are according to the Centrifuge[154] website:

1. The name of a genome, or the name corresponding to a taxonomic ID (the second column) at a rank higher than the strain.
2. The taxonomic ID.
3. The taxonomic rank.
4. The length of the genome sequence.
5. The number of reads classified to this genomic sequence including multi-classified reads.
6. The number of reads uniquely classified to this genomic sequence.
7. The proportion of this genome normalized by its genomic length.

Kraken-like report

If we would like to generate a report as generated with the former tool Kraken2[155], we can do it like this:

```
$ centrifuge-kreport -x p_compressed+h+v evolved-6-R1-results.txt > evolved-6-R1-kreport.txt
```

```
  0.00  0       0       U       0       unclassified
 78.74  163     0       -       1       root
 78.74  163     0       -       131567    cellular organisms
 78.74  163     0       D       2           Bacteria
 54.67  113     0       P       1224          Proteobacteria
 36.60  75      0       C       1236            Gammaproteobacteria
 31.18  64      0       O       91347             Enterobacterales
 30.96  64      0       F       543                 Enterobacteriaceae
 23.89  49      0       G       561                   Escherichia
 23.37  48      48      S       562                     Escherichia coli
  0.40  0       0       S       564                     Escherichia fergusonii
  0.12  0       0       S       208962                  Escherichia albertii
  3.26  6       0       G       570                   Klebsiella
  3.14  6       6       S       573                     Klebsiella pneumoniae
  0.12  0       0       S       548                     [Enterobacter] aerogenes
  2.92  6       0       G       620                   Shigella
  1.13  2       2       S       623                     Shigella flexneri
  0.82  1       1       S       624                     Shigella sonnei
  0.50  1       1       S       1813821                 Shigella sp. PAMC 28760
  0.38  0       0       S       621                     Shigella boydii
```

[153] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[154] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[155] https://www.ccb.jhu.edu/software/kraken2/

This gives a similar (not the same) report as the Kraken2[156] tool. The report is tab-delimited, with one line per taxon. The fields of the output, from left-to-right, are as follows:

1. Percentage of reads covered by the clade rooted at this taxon
2. Number of reads covered by the clade rooted at this taxon
3. Number of reads assigned directly to this taxon
4. A rank code, indicating (U)nclassified, (D)omain, (K)ingdom, (P)hylum, (C)lass, (O)rder, (F)amily, (G)enus, or (S)pecies. All other ranks are simply "-".
5. NCBI Taxonomy ID
6. The indented scientific name

6.6 Visualisation (Krona)

We use the Krona[157] tools to create a nice interactive visualisation of the taxa content of our sample [ONDOV2011]. Fig. 6.2 shows an example (albeit an artificial one) snapshot of the visualisation Krona[158] provides. Fig. 6.2 is a snapshot of the interactive web-page similar to the one we try to create.

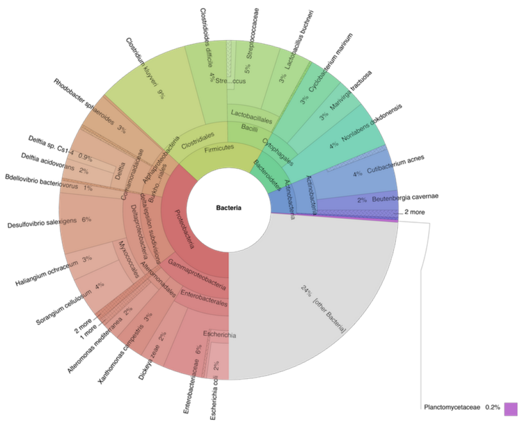


Fig. 6.2: Example of an Krona output webpage.

6.6.1 Installation

Install Krona[159] with:

```
$ conda create --yes -n krona krona
$ conda activate krona
```

First some house-keeping to make the Krona[160] installation work. Do not worry to much about what is happening here.

```
# we delete a symbolic link that is not correct
$ rm -rf ~/miniconda3/envs/ngs/opt/krona/taxonomy

# we create a directory in our home where the krona database will live
```

(continues on next page)

[156] https://www.ccb.jhu.edu/software/kraken2/
[157] https://github.com/marbl/Krona/wiki
[158] https://github.com/marbl/Krona/wiki
[159] https://github.com/marbl/Krona/wiki
[160] https://github.com/marbl/Krona/wiki

(continued from previous page)

```
$ mkdir -p ~/krona/taxonomy

# now we make a symbolic link to that directory
$ ln -s ~/krona/taxonomy ~/miniconda3/envs/ngs/opt/krona/taxonomy
```

6.6.2 Build the taxonomy

We need to build a taxonomy database for Krona[161]. However, if this fails we will skip this step and just download a pre-build one. Lets first try to build one.

```
$ ktUpdateTaxonomy.sh ~/krona/taxonomy
```

Attention: Should this fail we can download a pre-build database on the *Downloads* (page 77) page via a browser.

Once you have downloaded the file, follow these steps:

```
# we unzip the file
$ gzip -d taxonomy.tab.gz

# we move the unzipped file to our taxonomy directory we specified in the step before.
$ mv taxonomy.tab ~/krona/taxonomy
```

6.6.3 Visualise

Now, we use the tool `ktImportTaxonomy` from the Krona[162] tools to create the html web-page. We first need build a two column file (`read_id<tab>tax_id`) as input to the `ktImportTaxonomy` tool. We will do this by cutting the columns out of either the Kraken2[163] or Centrifuge[164] results:

```
# Kraken2
$ cd kraken
$ cat evol1.kraken | cut -f 2,3 > evol1.kraken.krona
$ ktImportTaxonomy evol1.kraken.krona
$ firefox taxonomy.krona.html

# Centrifuge
$ cd centrifuge
$ cat evol1-results.txt | cut -f 1,3 > evol1-results.krona
$ ktImportTaxonomy evol1-results.krona
$ firefox taxonomy.krona.html
```

What happens here is that we extract the second and third column from the Kraken2[165] results. Afterwards, we input these to the Krona[166] script, and open the resulting web-page in a bowser. Done!

[161] https://github.com/marbl/Krona/wiki
[162] https://github.com/marbl/Krona/wiki
[163] https://www.ccb.jhu.edu/software/kraken2/
[164] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[165] https://www.ccb.jhu.edu/software/kraken2/
[166] https://github.com/marbl/Krona/wiki

CHAPTER

SEVEN

VARIANT CALLING

7.1 Preface

In this section we will use our genome assembly based on the ancestor and call genetic variants in the evolved line [NIELSEN2011].

7.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 7.1.

7.3 Learning outcomes

After studying this tutorial section you should be able to:

#. Use tools to call variants based on a reference genome. #, Be able to describe what influences the calling of variants.

7.4 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
assembly/
data/
kraken/
mappings/
multiqc_data/
trimmed/
trimmed-fastqc/
```

Attention: If you have not run the previous section on *Read mapping* (page 25), you can download the mapped data needed for this section here: *Downloads* (page 77). Download the file to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O mappings.tar.gz https://osf.io/g5at8/download
tar xvzf mappings.tar.gz
```

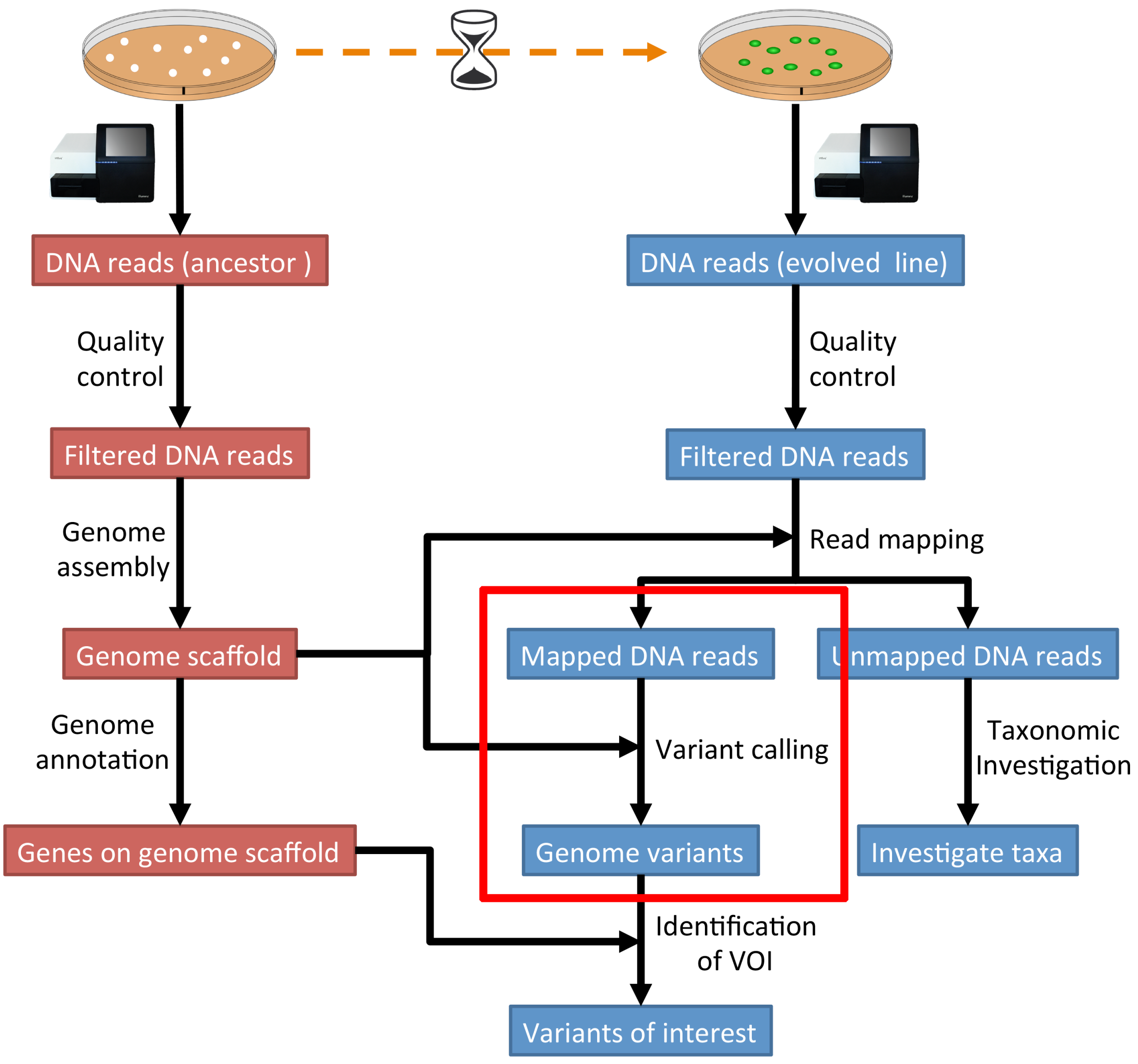


Fig. 7.1: The part of the workflow we will work on in this section marked in red.

7.5 Installing necessary software

Tools we are going to use in this section and how to intall them if you not have done it yet.

```
# activate the env
$ conda create --yes -n var samtools bamtools freebayes bedtools vcflib rtg-tools bcftools␣
↪matplotlib
$ conda activate var
```

7.6 Preprocessing

We first need to make an index of our reference genome as this is required by the SNP caller. Given a scaffold/contig file in fasta-format, e.g. `scaffolds.fasta` which is located in the directory `assembly/`, use SAMtools[171] to do this:

```
$ samtools faidx assembly/scaffolds.fasta
```

Furthermore we need to pre-process our mapping files a bit further and create a bam-index file (`.bai`) for the bam-file we want to work with:

```
$ bamtools index -in mappings/evol1.sorted.dedup.q20.bam
```

Lets also create a new directory for the variants:

```
$ mkdir variants
```

7.7 Calling variants

7.7.1 Freebayes

We can call variants with a tool called freebayes[172]. Given a reference genome scaffold file in fasta-format, e.g. `scaffolds.fasta` and the index in `.fai` format and a mapping file (.bam file) and a mapping index (.bai file), we can call variants with freebayes[173] like so:

```
# Now we call variants and pipe the results into a new file
$ freebayes -p 1 -f assembly/scaffolds.fasta mappings/evol1.sorted.dedup.q20.bam > variants/evol1.
↪freebayes.vcf
```

- `-p 1`: specifies the ploidy level. *E.Coli* are haploid.

7.8 Post-processing

7.8.1 Understanding the output files (.vcf)

Lets look at a vcf-file:

```
# first 10 lines, which are part of the header
$ cat variants/evol1.freebayes.vcf | head
```

[171] http://samtools.sourceforge.net/
[172] https://github.com/ekg/freebayes
[173] https://github.com/ekg/freebayes

```
##fileformat=VCFv4.2
##fileDate=20200122
##source=freeBayes v1.3.1-dirty
##reference=assembly/scaffolds.fasta
##contig=<ID=NODE_1_length_348724_cov_30.410613,length=348724>
##contig=<ID=NODE_2_length_327290_cov_30.828326,length=327290>
##contig=<ID=NODE_3_length_312063_cov_30.523209,length=312063>
##contig=<ID=NODE_4_length_202800_cov_31.500777,length=202800>
##contig=<ID=NODE_5_length_164027_cov_28.935175,length=164027>
##contig=<ID=NODE_6_length_144088_cov_29.907986,length=144088>
```

Lets look at the variants:

```
# remove header lines and look at top 4 entires
$ cat variants/evol1.freebayes.vcf | grep -v '##' | head -4
```

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  unknown
NODE_1_length_348724_cov_30.410613      375     .       A       C       0       .       AB=0;ABP=0;
↪AC=0;AF=0;AN=1;AO=3;CIGAR=1X;DP=21;DPB=21;DPRA=0;EPP=3.73412;EPPR=3.49285;GTI=0;LEN=1;MEANALT=1;
↪MQM=44;MQMR=40.3333;NS=1;NUMALT=1;ODDS=63.5226;PAIRED=1;PAIREDR=1;PAO=0;PQA=0;PQR=0;PRO=0;QA=53;
↪QR=414;RO=18;RPL=2;RPP=3.73412;RPPR=7.35324;RPR=1;RUN=1;SAF=3;SAP=9.52472;SAR=0;SRF=14;SRP=15.074;
↪SRR=4;TYPE=snp       GT:DP:AD:RO:QR:AO:QA:GL      0:21:18,3:18:414:3:53:0,-29.6927
NODE_1_length_348724_cov_30.410613      393     .       T       A       0       .       AB=0;ABP=0;
↪AC=0;AF=0;AN=1;AO=2;CIGAR=1X;DP=24;DPB=24;DPRA=0;EPP=7.35324;EPPR=6.56362;GTI=0;LEN=1;MEANALT=1;
↪MQM=36;MQMR=42.9545;NS=1;NUMALT=1;ODDS=127.074;PAIRED=1;PAIREDR=1;PAO=0;PQA=0;PQR=0;PRO=0;QA=21;
↪QR=717;RO=22;RPL=2;RPP=7.35324;RPPR=3.0103;RPR=0;RUN=1;SAF=2;SAP=7.35324;SAR=0;SRF=17;SRP=17.2236;
↪SRR=5;TYPE=snp       GT:DP:AD:RO:QR:AO:QA:GL      0:24:22,2:22:717:2:21:0,-57.4754
NODE_1_length_348724_cov_30.410613      612     .       A       C       2.32041e-15     .       ␣
↪AB=0;ABP=0;AC=0;AF=0;AN=1;AO=3;CIGAR=1X;DP=48;DPB=48;DPRA=0;EPP=9.52472;EPPR=11.1654;GTI=0;LEN=1;
↪MEANALT=1;MQM=60;MQMR=60;NS=1;NUMALT=1;ODDS=296.374;PAIRED=1;PAIREDR=0.977778;PAO=0;PQA=0;PQR=0;
↪PRO=0;QA=53;QR=1495;RO=45;RPL=0;RPP=9.52472;RPPR=3.44459;RPR=3;RUN=1;SAF=3;SAP=9.52472;SAR=0;
↪SRF=19;SRP=5.37479;SRR=26;TYPE=snp  GT:DP:AD:RO:QR:AO:QA:GL      0:48:45,3:45:1495:3:53:0,-129.869
```

The fields in a vcf-file are described in he table (Table 7.1) below:

Table 7.1: The vcf-file format fields.

Col	Field	Description
1	CHROM	Chromosome name
2	POS	1-based position. For an indel, this is the position preceding the indel.
3	ID	Variant identifier. Usually the dbSNP rsID.
4	REF	Reference sequence at POS involved in the variant. For a SNP, it is a single base.
5	ALT	Comma delimited list of alternative seuqence(s).
6	QUAL	Phred-scaled probability of all samples being homozygous reference.
7	FILTER	Semicolon delimited list of filters that the variant fails to pass.
8	INFO	Semicolon delimited list of variant information.
9	FORMAT	Colon delimited list of the format of individual genotypes in the following fields.
10+	Sample(s)	Individual genotype information defined by FORMAT.

7.8.2 Statistics

Now we can use it to do some statistics and filter our variant calls.

First, to prepare out vcf-file for querying we need to index it with `tabix`:

```
# compress file
$ bgzip variants/evol1.freebayes.vcf
# index
$ tabix -p vcf variants/evol1.freebayes.vcf.gz
```

- `-p vcf`: input format

We can get some quick stats with `rtg vcfstats`:

```
$ rtg vcfstats variants/evol1.freebayes.vcf.gz
```

Example output from `rtg vcfstats`:

```
Location                     : variants/evol1.freebayes.vcf.gz
Failed Filters               : 0
Passed Filters               : 35233
SNPs                         : 55
MNPs                         : 6
Insertions                   : 3
Deletions                    : 5
Indels                       : 0
Same as reference            : 35164
SNP Transitions/Transversions: 0.83 (25/30)
Total Haploid                : 69
Haploid SNPs                 : 55
Haploid MNPs                 : 6
Haploid Insertions           : 3
Haploid Deletions            : 5
Haploid Indels               : 0
Insertion/Deletion ratio     : 0.60 (3/5)
Indel/SNP+MNP ratio          : 0.13 (8/61)
```

However, we can also run BCFtools[174] to extract more detailed statistics about our variant calls:

```
$ bcftools stats -F assembly/scaffolds.fasta -s - variants/evol1.freebayes.vcf.gz > variants/evol1.
↪freebayes.vcf.gz.stats
```

- `-s -`: list of samples for sample stats, "-" to include all samples
- `-F FILE`: faidx indexed reference sequence file to determine INDEL context

Now we take the stats and make some plots (e.g. Fig. 7.2) which are particular of interest if having multiple samples, as one can easily compare them. However, we are only working with one here:

```
$ mkdir variants/plots
$ plot-vcfstats -p variants/plots/ variants/evol1.freebayes.vcf.gz.stats
```

- `-p`: The output files prefix, add a slash at the end to create a new directory.

[174] http://www.htslib.org/doc/bcftools.html

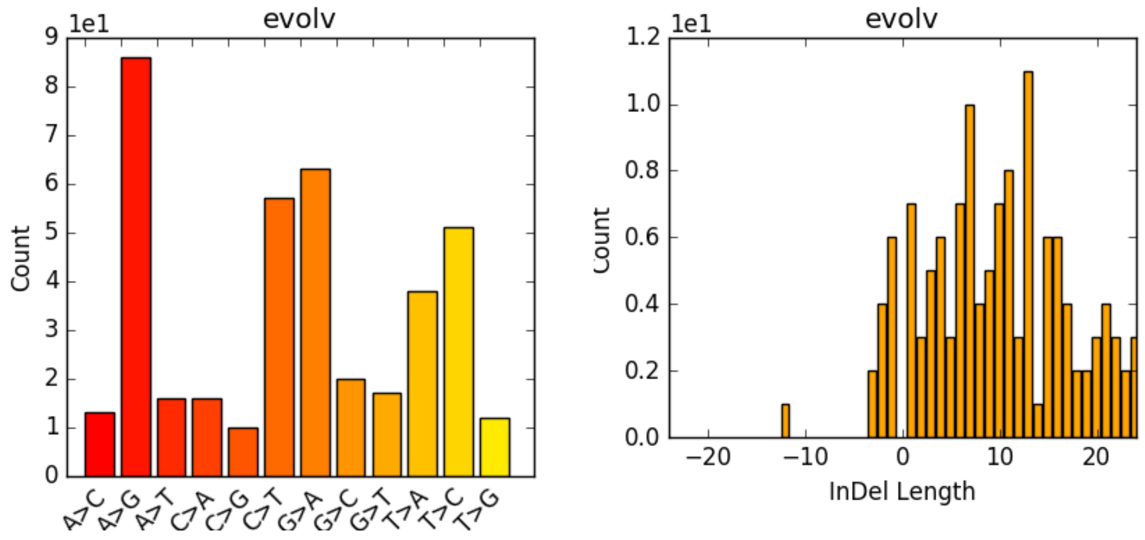


Fig. 7.2: Example of `plot-vcfstats` output.

7.8.3 Variant filtration

Variant filtration is a big topic in itself [OLSEN2015]. There is no consens yet and research on how to best filter variants is ongoing.

We will do some simple filtration procedures here. For one, we can filter out low quality reads.

Here, we only include variants that have quality > 30.

```
# use rtg vcfffilter
$ rtg vcffilter -q 30 -i variants/evol1.freebayes.vcf.gz -o variants/evol1.freebayes.q30.vcf.gz
```

- `-i FILE`: input file
- `-o FILE`: output file
- `-q FLOAT`: minimal allowed quality in output.

or use vcflib[175]:

```
# or use vcflib
$ zcat variants/evol1.freebayes.vcf.gz  | vcffilter -f "QUAL >= 30" | gzip > variants/evol1.
↪freebayes.q30.vcf.gz
```

- `-f "QUAL >= 30"`: we only include variants that have been called with quality >= 30.

Quick stats for the filtered variants:

```
# look at stats for filtered
$ rtg vcfstats variants/evol1.freebayes.q30.vcf.gz
```

freebayes[176] adds some extra information to the vcf-files it creates. This allows for some more detailed filtering. This strategy will NOT work on calls done with e.g. SAMtools[177]/bcftools mpileup called variants. Here we filter, based on some recommendation form the developer of freebayes[178]:

```
$ zcat variants/evol1.freebayes.vcf.gz | vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR >␣
↪0 & RPR > 1 & RPL > 1" | bgzip > variants/evol1.freebayes.filtered.vcf.gz
```

[175] https://github.com/vcflib/vcflib#vcflib
[176] https://github.com/ekg/freebayes
[177] http://samtools.sourceforge.net/
[178] https://github.com/ekg/freebayes

- `QUAL > 1`: removes really bad sites
- `QUAL / AO > 10`: additional contribution of each obs should be 10 log units (~ Q10 per read)
- `SAF > 0 & SAR > 0`: reads on both strands
- `RPR > 1 & RPL > 1`: at least two reads "balanced" to each side of the site

```
$ tabix -p vcf variants/evol1.freebayes.filtered.vcf.gz
```

This strategy used here will do for our purposes. However, several more elaborate filtering strategies have been explored, e.g. here[179].

Todo: Look at the statistics. One ratio that is mentioned in the statistics is transition transversion ratio (*ts/tv*). Explain what this ratio is and why the observed ratio makes sense.

Todo: Call and filter variants for the second evolved strain, similarily to what ws described here for the first strain. Should you be unable to do it, check the code section: *Code: Variant calling* (page 75).

[179] https://github.com/ekg/freebayes#observation-filters-and-qualities

CHAPTER

EIGHT

GENOME ANNOTATION

8.1 Preface

In this section you will predict genes and assess your assembly using Augustus[182] and BUSCO[183], as well as Prokka[184].

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

8.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 8.1.

8.3 Learning outcomes

After studying this section of the tutorial you should be able to:

1. Explain how annotation completeness is assessed using orthologues
2. Use bioinformatics tools to perform gene prediction
3. Use genome-viewing software to graphically explore genome annotations and NGS data overlays

8.4 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
assembly/
data/
kraken/
mappings/
multiqc_data/
trimmed/
trimmed-fastqc/
variants/
```

[182] http://augustus.gobics.de
[183] http://busco.ezlab.org
[184] https://github.com/tseemann/prokka

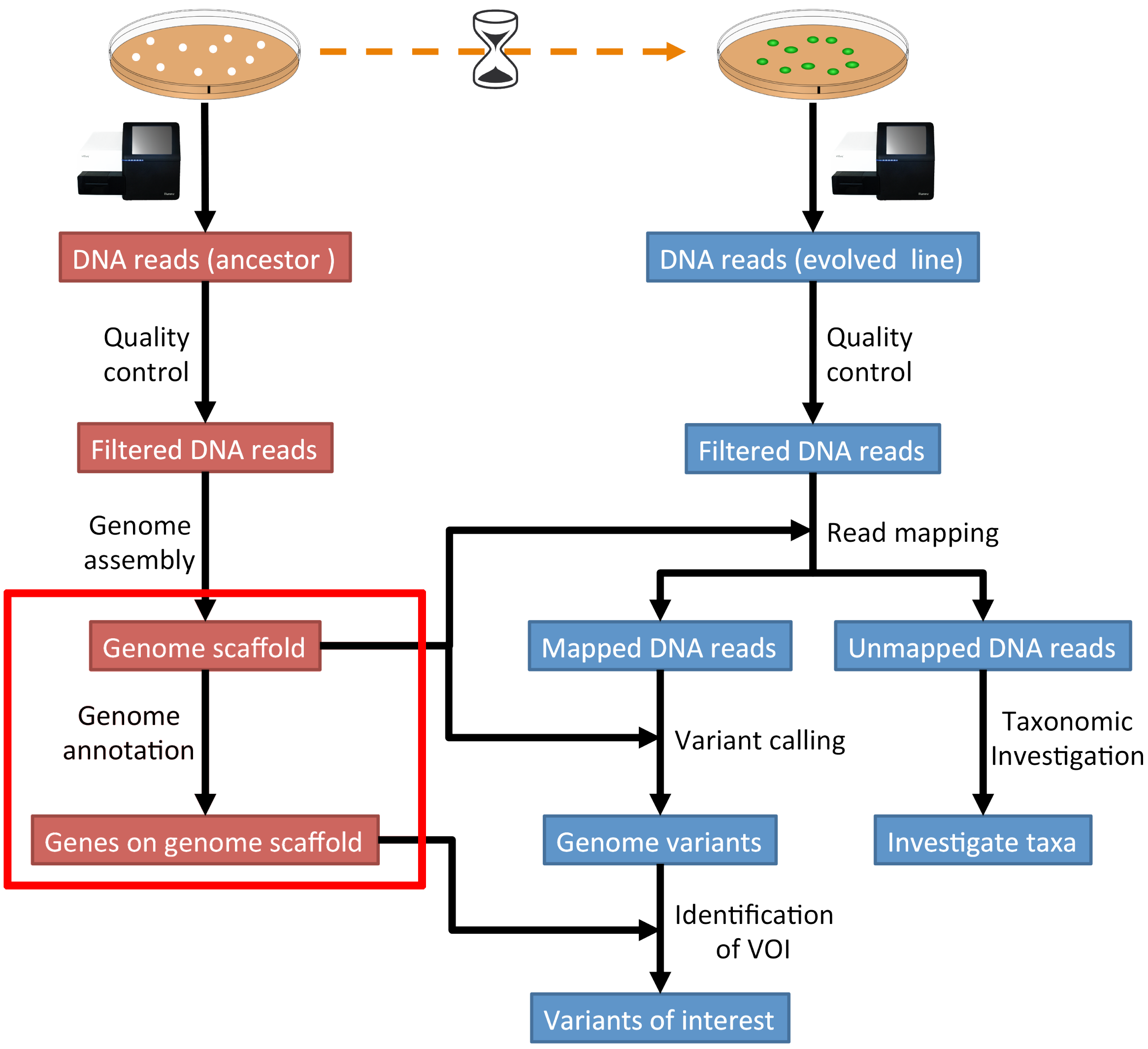


Fig. 8.1: The part of the workflow we will work on in this section marked in red.

Attention: If you have not run the previous section *Genome assembly* (page 19), you can download the genome assembly needed for this section here: *Downloads* (page 77). Download the file to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O assembly.tar.gz https://osf.io/t2zpm/download
tar xvzf assembly.tar.gz
```

8.5 Installing the software

```
# activate the env
$ conda create --yes -n anno busco
```

This will install both the Augustus[185] [STANKE2005] and the BUSCO[186] [SIMAO2015] software, which we will use (separately) for gene prediction and assessment of assembly completeness, respectively.

Make a directory for the annotation results:

```
$ mkdir annotation
$ cd annotation
```

We need to get the database that BUSCO[187] will use to assess orthologue presence absence in our genome annotation. BUSCO[188] provides a command to list all available datasets and download datasets.

```
$ busco --list-datasets
```

```
INFO:   Downloading information on latest versions of BUSCO data...

################################################

Datasets available to be used with BUSCOv4 as of 2019/11/27:

bacteria_odb10
    - acidobacteria_odb10
    - actinobacteria_phylum_odb10
        - actinobacteria_class_odb10
            - corynebacteriales_odb10
            - micrococcales_odb10
            - propionibacteriales_odb10
            - streptomycetales_odb10
            - streptosporangiales_odb10
        - coriobacteriia_odb10
            - coriobacteriales_odb10
...
```

BUSCO[189] will download the dataset when starting an analysis.

We also need to place the configuration file for this program in a location in which we have "write" privileges. Do this recursively for the entire config directory, placing it into your current annotation directory:

```
$ cp -r ~/miniconda3/envs/anno/config/ .
```

[185] http://augustus.gobics.de
[186] http://busco.ezlab.org
[187] http://busco.ezlab.org
[188] http://busco.ezlab.org
[189] http://busco.ezlab.org

8.6 Assessment of orthologue presence and absence

BUSCO[190] will assess orthologue presence absence using blastn[191], a rapid method of finding close matches in large databases (we will discuss this in lecture). It uses blastn[192] to make sure that it does not miss any part of any possible coding sequences. To run the program, we give it

- A fasta format input file
- A name for the output files
- The name of the lineage database against which we are assessing orthologue presence absence (that we downloaded above)
- An indication of the type of annotation we are doing (genomic, as opposed to transcriptomic or previously annotated protein files).
- The config file to use

```
$ busco  -i ../assembly/scaffolds.fasta -o my_anno -l bacteria_odb10 -m geno --config config/config.
↪ini
```

Navigate into the output directory you created. There are many directories and files in there containing information on the orthologues that were found, but here we are only really interested in one: the summary statistics. This is located in the `short_summary*.txt` file. Look at this file. It will note the total number of orthologues found, the number expected, and the number missing. This gives an indication of your genome completeness.

Todo: Is it necessarily true that your assembly is incomplete if it is missing some orthologues? Why or why not?

8.7 Annotation with Augustus[193]

We will use Augustus[194] to perform gene prediction. This program implements a hidden markov model (HMM) to infer where genes lie in the assembly you have made. To run the program you need to give it:

- Information as to whether you would like the genes called on both strands (or just the forward or reverse strands)
- A "model" organism on which it can base it's HMM parameters on (in this case we will use E.coli)
- The location of the assembly file
- A name for the output file, which will be a .gff (general feature format) file.
- We will also tell it to display a progress bar as it moves through the genome assembly.

```
$ augustus --progress=true --strand=both --species=E_coli_K12 --AUGUSTUS_CONFIG_PATH=config ../
↪assembly/scaffolds.fasta > augustus.gff
```

Note: Should the process of producing your annotation fail, you can download a annotation manually from *Downloads* (page 77). Remember to unzip the file.

[190] http://busco.ezlab.org
[191] https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch
[192] https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch
[193] http://augustus.gobics.de
[194] http://augustus.gobics.de

8.8 Annotation with Prokka[195]

Install Prokka[196]:

```
$ conda create --yes -n prokka prokka
$ conda activate prokka
```

Run Prokka[197]:

```
$ prokka --kingdom Bacteria --genus Escherichia --species coli --outdir annotation assembly/
↪scaffolds.fasta
```

Your results will be in the `annotation` directory with the prefix `PROKKA`.

8.9 Interactive viewing

We will use the software IGV[198] to view the assembly, the gene predictions you have made, and the variants that you have called, all in one window.

8.9.1 IGV[199]

```
$ conda activate anno
$ conda install --yes igv
```

To run IGV type:

```
$ igv
```

This will open up a new window. Navigate to that window and open up your genome assembly:

- **Genomes** -> **Load Genome from File**
- Load your assembly (`scaffolds.fasta`), not your gff file.

Load the tracks:

- **File** -> **Load from File**
- Load your unzipped `vcf` file from section: *Variant calling* (page 47)
- Load your unzipped `gff` file from this section.

At this point you should be able to zoom in and out to see regions in which there are SNPs or other types of variants. You can also see the predicted genes. If you zoom in far enough, you can see the sequence (DNA and protein).

If you have time and interest, you can right click on the sequence and copy it. Open a new browser window and go to the blastn homepage. There, you can blast your gene of interest (GOI) and see if blast can assign a function to it.

The end goal of this lab will be for you to select a variant that you feel is interesting (e.g. due to the gene it falls near or within), and hypothesize as to why that mutation might have increased in frequency in these evolving populations.

[195] https://github.com/tseemann/prokka
[196] https://github.com/tseemann/prokka
[197] https://github.com/tseemann/prokka
[198] http://software.broadinstitute.org/software/igv/
[199] http://software.broadinstitute.org/software/igv/

CHAPTER

NINE

ORTHOLOGY AND PHYLOGENY

9.1 Preface

In this section you will use some software to find orthologue genes and do phylogenetic reconstructions.

9.2 Learning outcomes

After studying this tutorial you should be able to:

1. Use bioinformatics software to find orthologues in the NCBI database.
2. Use bioinformatics software to perform sequence alignment.
3. Use bioinformatics software to perform phylogenetic reconstructions.

9.3 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
annotation/
assembly/
data/
kraken/
mappings/
multiqc_data/
trimmed/
trimmed-fastqc/
variants/
```

Make a directory for the phylogeny results (in your analysis directory):

```
$ mkdir phylogeny
```

9.4 Installing the software

```
$ conda create -n phylo blast mafft raxml iqtree bedtools
```

This will install a BLAST[202] executable that you can use to remotely query the NCBI database and the MAFFT[203] alignment program that you can use to align sequences. We also install RAxML[204] and IQ-TREE[205], phylogenetic tree inference tools, which use maximum-likelihood (ML) optimality criterion.

9.5 The gene we are using

We are using the the gene *gnd* (gluconate-6-phosphate dehydrogenase) as an example. *gnd* is a highly polymorphic gene within E. coli populations, likely due to interstrain transfer and recombination. This may be a result of its proximity to the *rfb* region, which determines O antigen structure.

First, we are going to make a bed-file to get coordinates from file:

```
# Figure out the chromosome and location of the `gnd` gene
# there *must* be a better way to do this
grep -B 1 ‘NODE_\|gnd’ ../annotation/*gbk

# open a file to put the coordinates into using vi or nano
vi gnd.bed

# these are the coordinates from the contigs-file (Yours might be different), we copy into the vi/
↪nano buffer
NODE_42_length_35862_cov_7.082632   625     2031

# safe the file and exit vi/nano
```

Hint: To edit in vi editor, you will need to press the escape key and “a” or “e”. To save in vi, you will need to press the escape key and “w” (write). To quit vi, you will need to press the escape key and “q” (quit).

Use bedtools to extract the nucleotide sequence for the region:

```
bedtools getfasta -fi ../assembly/contigs.fasta -bed gnd.bed > gnd.fasta
```

Now, we have a fasta-file with exactly on genic region, the one from the *gnd* gene.

9.6 Finding orthologues using BLAST

We will use a remote BLAST[206] of ou gene against the *nr*-database:

```
blastn -db nt -query gnd.fasta -remote -evalue 1e-100 -outfmt "6 qseqid sseqid sseq" > gnd_blast_
↪hits.out
```

Some of the arguments explained:

- -db: The name of the database that we are BLASTing against
- -query: A fasta format input file

[202] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[203] https://mafft.cbrc.jp/alignment/software/
[204] https://github.com/stamatak/standard-RAxML
[205] http://www.iqtree.org/
[206] https://blast.ncbi.nlm.nih.gov/Blast.cgi

- `-outfmt "6 qseqid sseqid sseq"`: Some notes about the format we want
- `-evalue 1e-100`: An evalue cutoff for inclusion of results

Next, we are formating the result into fasta-format using the program `awk`:

```
awk 'BEGIN { OFS = "\n" } { print ">"$2, $3 }' gnd_blast_hits.out > gnd_blast_hits.fasta
```

Append the fasta file of your E. coli gene to this file, using whatever set of commands you wish/know. For example:

```
# append the gnd gene from our E. coli!
cat gnd.fasta >> gnd_blast_hits.fasta
```

9.7 Performing an alignment

We will use MAFFT[207] to perform our alignment on all the sequences in the BLAST[208] fasta file. This syntax is very simple (change the filenames accordingly):

```
$ mafft gnd_blast_hits.fasta > gnd_blast_hits.aln
```

9.8 Building a phylogeny

We will use RAxML[209] to build our phylogeny. This uses a maximum likelihood method to infer parameters of evolution and the topology of the tree. Again, the syntax of the command is fairly simple, except you must make sure that you are using the directory in which RAxML[210] sits.

The arguments are:

- `-s`: an alignment file
- `-m`: a model of evolution. In this case we will use a general time reversible model with gamma distributed rates (GTR+GAMMA)
- `-n`: outfile-name
- `-p`: specify a random number seed for the parsimony inferences

```
$ raxmlHPC -s gnd_blast_hits.aln -m GTRGAMMA -n ecoli_tree -p 12345
```

We can also use IQ-TREE[211], which provides more information than RAxML[212].

```
iqtree -s gnd_blast_hits.aln
```

[207] https://mafft.cbrc.jp/alignment/software/
[208] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[209] https://github.com/stamatak/standard-RAxML
[210] https://github.com/stamatak/standard-RAxML
[211] http://www.iqtree.org/
[212] https://github.com/stamatak/standard-RAxML

9.9 Visualizing the phylogeny

We will use the online software Interactive Tree of Life (iTOL)[213] to visualize the tree. Navigate to this homepage. Open the file containing your tree (`*bestTree.out`), copy the contents, and paste into the web page (in the Tree text box).

You should then be able to zoom in and out to see where your taxa is. To find out the closest relative, you will have to use the NCBI taxa page[214].

[213] http://itol.embl.de/upload.cgi
[214] https://www.ncbi.nlm.nih.gov/Taxonomy/TaxIdentifier/tax_identifier.cgi

CHAPTER

TEN

VARIANTS-OF-INTEREST

10.1 Preface

In this section we will use our genome annotation of our reference and our genome variants in the evolved line to find variants that are interesting in terms of the observed biology.

Note: You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

10.2 Overview

The part of the workflow we will work on in this section can be viewed in Fig. 10.1.

10.3 Learning outcomes

After studying this section of the tutorial you should be able to:

1. Identify variants of interests.
2. Understand how the variants might affect the observed biology in the evolved line.

10.4 Before we start

Lets see how our directory structure looks so far:

```
$ cd ~/analysis
$ ls -1F
```

```
annotation/
assembly/
data/
kraken/
mappings/
phylogeny/
trimmed/
trimmed-fastqc/
variants/
```

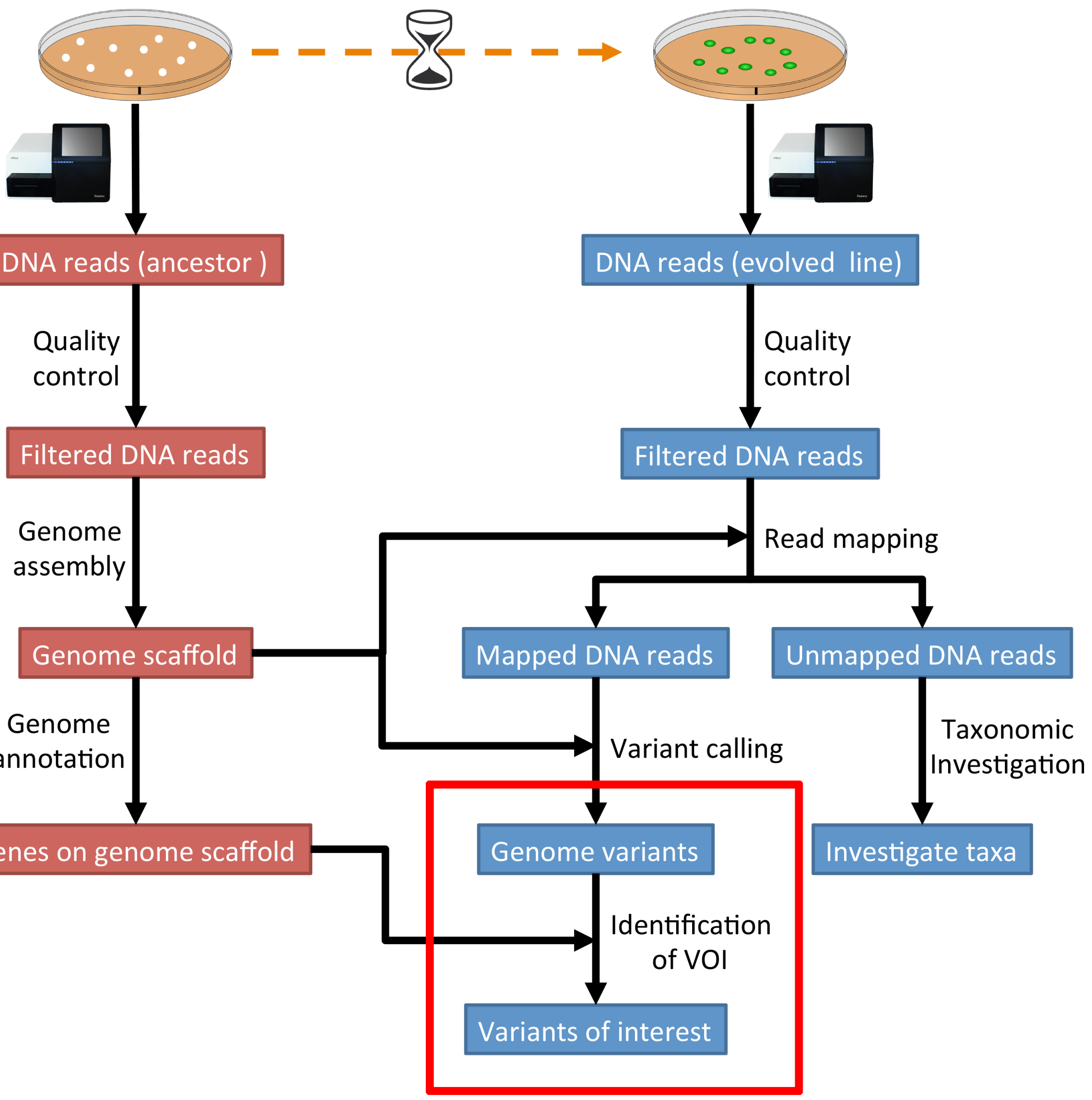


Fig. 10.1: The part of the workflow we will work on in this section marked in red.

Attention: If you have not run the previous sections on *Genome assembly* (page 19) and *Variant calling* (page 47), you can download the variant calls and the genome assembly needed for this section here: *Downloads* (page 77). Download the files to the `~/analysis` directory and decompress. Alternatively on the CLI try:

```
cd ~/analysis
wget -O assembly.tar.gz https://osf.io/t2zpm/download
tar xvzf assembly.tar.gz
eget -O variants.tar.gz https://osf.io/4nzrm/download
tar xvzf variants.tar.gz
```

10.5 General comments for identifying variants-of-interest

Things to consider when looking for variants-of-interest:

- The quality score of the variant call.
  - Do we call the variant with a higher then normal score?
- The mapping quality score.
  - How confident are we that the reads were mapped at the position correctly?
- The location of the SNP.
  - SNPs in larger contigs are probably more interesting than in tiny contigs.
  - Does the SNP overlap a coding region in the genome annotation?
- The type of SNP.
  - substitutions vs. indels

10.6 SnpEff

We will be using SnpEff[215] to annotate our identified variants. The tool will tell us on to which genes we should focus further analyses.

10.6.1 Installing software

Tools we are going to use in this section and how to install them if you not have done it yet:

```
$ conda create -n voi snpeff genometools-genometools bedtools
```

Make a directory for the results (in your analysis directory) and change into the directory:

```
$ mkdir voi

# change into the directory
$ cd voi
```

[215] http://snpeff.sourceforge.net/index.html

10.6.2 Prepare SnpEff database

We need to create our own config-file for SnpEff[216]. Where is the `snpEff.config`:

```
$ find ~ -name snpEff.config
/home/guest/miniconda3/envs/voi/share/snpeff-4.3.1t-3/snpEff.config
```

This will give you the path to the `snpEff.config`. It might be looking a bit different then the one shown here, depending on the version of SnpEff[217] that is installed.

Make a local copy of the `snpEff.config` and then edit it with an editor of your choice:

```
$ cp /home/guest/miniconda3/envs/voi/share/snpeff-4.3.1t-3/snpEff.config .
$ nano snpEff.config
```

Make sure the data directory path in the `snpEff.config` looks like this:

```
data.dir = ./data/
```

There is a section with databases, which starts like this:

```
#-------------------------------------------------------------------------------
# Databases & Genomes
#
# One entry per genome version.
#
# For genome version 'ZZZ' the entries look like
#   ZZZ.genome              : Real name for ZZZ (e.g. 'Human')
#   ZZZ.reference           : [Optional] Comma separated list of URL to site/s Where information␣
↪for building ZZZ database was extracted.
#   ZZZ.chrName.codonTable  : [Optional] Define codon table used for chromosome 'chrName' (Default:
↪'codon.Standard')
#
#-------------------------------------------------------------------------------
```

Add the following two lines in the database section underneath these header lines:

```
# my genome
mygenome.genome : EColiMut
```

Now, we need to create a local data folder called `./data/mygenome`.

```
# create folders
$ mkdir -p ./data/mygenome
```

Copy our genome assembly to the newly created data folder. The name needs to be `sequences.fa` or `mygenome.fa`:

```
$ cp ../assembly/scaffolds.fasta ./data/mygenome/sequences.fa
$ gzip ./data/mygenome/sequences.fa
```

Copy our genome annotation to the data folder. The name needs to be `genes.gff` (or `genes.gtf` for gtf-files).

```
$ cp ../annotation/PROKKA_12345.gff ./data/mygenome/genes.gff
$ gzip ./data/mygenome/genes.gff
```

Now we can build a new SnpEff[218] database:

[216] http://snpeff.sourceforge.net/index.html
[217] http://snpeff.sourceforge.net/index.html
[218] http://snpeff.sourceforge.net/index.html

```
$ snpEff build -c snpEff.config -gff3 -v mygenome > snpEff.stdout 2> snpEff.stderr
```

Note: Should this fail, due to gff-format of the annotation, we can try to convert the gff to gtf:

```
# using genometools
$ gt gff3_to_gtf ../annotation/PROKKA_12345.gff -o ./data/mygenome/genes.gtf
$ gzip ./data/mygenome/genes.gtf
```

Now, we can use the gtf annotation top build the database:

```
$ snpEff build -c snpEff.config -gtf22 -v mygenome > snpEff.stdout 2> snpEff.stderr
```

10.6.3 SNP annotation

Now we can use our new SnpEff[219] database to annotate some variants, e.g.:

```
$ snpEff -c snpEff.config mygenome ../variants/evol1.freebayes.filtered.vcf > evol1.freebayes.
↪filtered.anno.vcf
$ snpEff -c snpEff.config mygenome ../variants/evol2.freebayes.filtered.vcf > evol2.freebayes.
↪filtered.anno.vcf
```

SnpEff[220] adds ANN fields to the vcf-file entries that explain the effect of the variant.

Note: If you are unable to do the annotation, you can download an annotated vcf-file from *Downloads* (page 77).

10.6.4 Example

Lets look at one entry from the original vcf-file and the annotated one. We are only interested in the 8th column, which contains information regarding the variant. SnpEff[221] will add fields here :

```
# evol2.freebayes.filtered.vcf (the original), column 8
AB=0;ABP=0;AC=1;AF=1;AN=1;AO=37;CIGAR=1X;DP=37;DPB=37;DPRA=0;EPP=10.1116;EPPR=0;GTI=0;LEN=1;
↪MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=226.923;PAIRED=0.972973;PAIREDR=0;PAO=0;PQA=0;PQR=0;
↪PRO=0;QA=1155;QR=0;RO=0;RPL=12;RPP=12.9286;RPPR=0;RPR=25;RUN=1;SAF=26;SAP=16.2152;SAR=11;SRF=0;
↪SRP=0;SRR=0;TYPE=snp

# evol2.freebayes.filtered.anno.vcf, column 8
AB=0;ABP=0;AC=1;AF=1;AN=1;AO=37;CIGAR=1X;DP=37;DPB=37;DPRA=0;EPP=10.1116;EPPR=0;GTI=0;LEN=1;
↪MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=226.923;PAIRED=0.972973;PAIREDR=0;PAO=0;PQA=0;PQR=0;
↪PRO=0;QA=1155;QR=0;RO=0;RPL=12;RPP=12.9286;RPPR=0;RPR=25;RUN=1;SAF=26;SAP=16.2152;SAR=11;SRF=0;
↪SRP=0;SRR=0;TYPE=snp;ANN=T|missense_variant|MODERATE|HGGMJBFA_02792|GENE_HGGMJBFA_
↪02792|transcript|TRANSCRIPT_HGGMJBFA_02792|protein_coding|1/1|c.773G>A|p.Arg258His|773/1092|773/
↪1092|258/363||WARNING_TRANSCRIPT_NO_START_CODON,T|upstream_gene_variant|MODIFIER|HGGMJBFA_
↪02789|GENE_HGGMJBFA_02789|transcript|TRANSCRIPT_HGGMJBFA_02789|protein_coding||c.-4878G>
↪A||||4878|,T|upstream_gene_variant|MODIFIER|HGGMJBFA_02790|GENE_HGGMJBFA_
↪02790|transcript|TRANSCRIPT_HGGMJBFA_02790|protein_coding||c.-3568G>A||||3568|,T|upstream_gene_
↪variant|MODIFIER|HGGMJBFA_02791|GENE_HGGMJBFA_02791|transcript|TRANSCRIPT_HGGMJBFA_02791|protein_
↪coding||c.-442G>A||||442|,T|upstream_gene_variant|MODIFIER|HGGMJBFA_02794|GENE_HGGMJBFA_
↪02794|transcript|TRANSCRIPT_HGGMJBFA_02794|protein_coding||c.-1864C>T||||1864|,T|upstream_gene_
↪variant|MODIFIER|HGGMJBFA_02795|GENE_HGGMJBFA_02795|transcript|TRANSCRIPT_HGGMJBFA_02795|protein_
↪coding||c.-3530C>T||||3530|,T|upstream_gene_variant|MODIFIER|HGGMJBFA_02796|GENE_HGGMJBFA_
↪02796|transcript|TRANSCRIPT_HGGMJBFA_02796|protein_coding||c.-4492C>T||||4492|,T|downstream_gene_
↪variant|MODIFIER|HGGMJBFA_02793|GENE_HGGMJBFA_02793|transcript|TRANSCRIPT_HGGMJBFA_02793|protein_
↪coding||c.*840G>A||||840|
```

(continues on next page)

[219] http://snpeff.sourceforge.net/index.html
[220] http://snpeff.sourceforge.net/index.html
[221] http://snpeff.sourceforge.net/index.html

(continued from previous page)

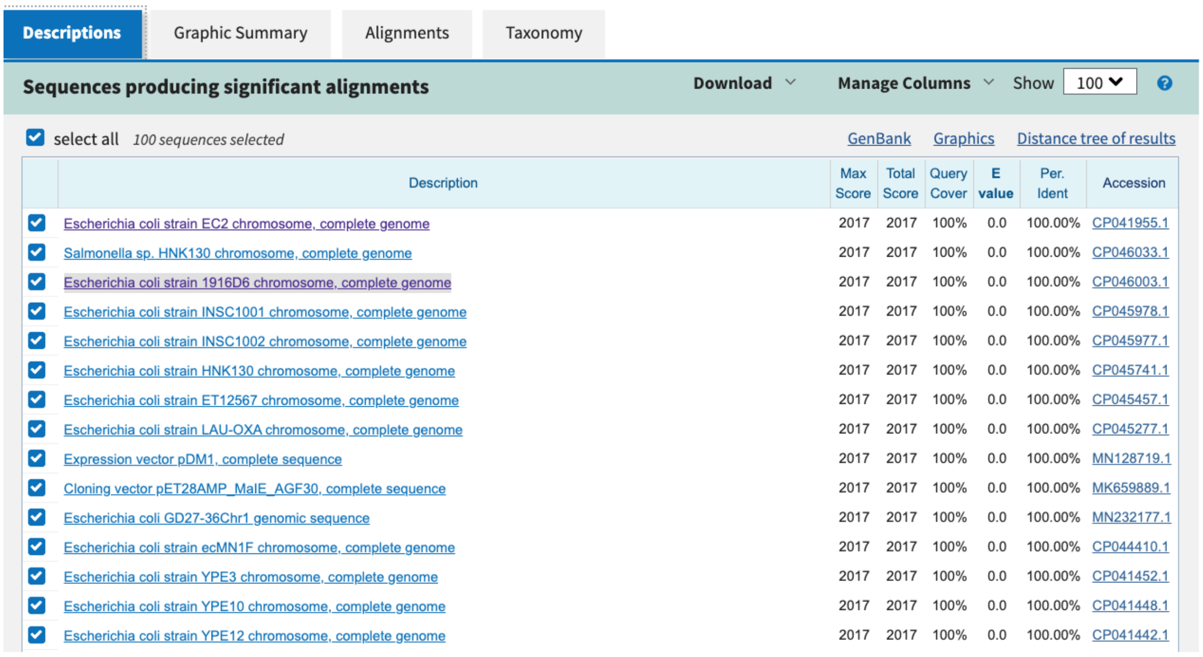
```
```

When expecting the second entry, we find that SnpEff[222] added annotation information starting with `ANN=T|missense_variant|....` If we look a bit more closely we find that the variant results in a amino acid change from a arginine to a histidine (`c.773G>A|p.Arg258His`). The codon for arginine is `CGN` and for histidine is `CAT/CAC`, so the variant in the second nucleotide of the codon made the amino acid change.

A quick BLAST[223] search of the CDS sequence, where the variant was found (extracted from the `genes.gff.gz`) shows that the closest hit is a DNA-binding transcriptional regulator from several different *E.Coli* strains.

```
# decompress annotation and genome
$ gzip -d data/mygenome/genes.gff.gz
$ gzip -d data/mygenome/sequences.fa.gz

# extract genes sequences
$ bedtools getfasta -fi data/mygenome/sequences.fa -bed data/mygenome/genes.gff > data/mygenome/
↪genes.fa
```



	Description	Max Score	Total Score	Query Cover	E value	Per. Ident	Accession
✓	Escherichia coli strain EC2 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP041955.1
✓	Salmonella sp. HNK130 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP046033.1
✓	Escherichia coli strain 1916D6 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP046003.1
✓	Escherichia coli strain INSC1001 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP045978.1
✓	Escherichia coli strain INSC1002 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP045977.1
✓	Escherichia coli strain HNK130 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP045741.1
✓	Escherichia coli strain ET12567 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP045457.1
✓	Escherichia coli strain LAU-OXA chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP045277.1
✓	Expression vector pDM1, complete sequence	2017	2017	100%	0.0	100.00%	MN128719.1
✓	Cloning vector pET28AMP_MalE_AGF30, complete sequence	2017	2017	100%	0.0	100.00%	MK659889.1
✓	Escherichia coli GD27-36Chr1 genomic sequence	2017	2017	100%	0.0	100.00%	MN232177.1
✓	Escherichia coli strain ecMN1F chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP044410.1
✓	Escherichia coli strain YPE3 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP041452.1
✓	Escherichia coli strain YPE10 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP041448.1
✓	Escherichia coli strain YPE12 chromosome, complete genome	2017	2017	100%	0.0	100.00%	CP041442.1

Fig. 10.2: Results of a BLAST[224] search of the CDS.

[222] http://snpeff.sourceforge.net/index.html
[223] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[224] https://blast.ncbi.nlm.nih.gov/Blast.cgi

CHAPTER

ELEVEN

QUICK COMMAND REFERENCE

11.1 Shell commands

```
# Where in the directory tree am I?
pwd

# List the documents and sub-directories in the current directory
ls

# a bit nicer listing with more information
ls -laF

# Change into your home directory
cd ~

# Change back into the last directory
cd -

# Change one directory up in the tree
cd ..

# Change explicitly into a directory "temp"
cd temp

# Quickly show content of a file "temp.txt"
# exist the view with "q", navigate line up and down with "k" and "j"
less temp.text

# Show the beginning of a file "temp.txt"
head temp.txt

# Show the end of a file "temp.txt"
tail temp.txt
```

11.2 General conda commands

```
# To update all packages
conda update --all --yes

# List all packages installed
conda list [-n env]

# conda list environments
conda env list

```

(continues on next page)

(continued from previous page)

```
# create new env
conda create -n [name] package [package] ...

# activate env
conda activate [name]

# deavtivate env
conda deactivate
```

CHAPTER

TWELVE

CODING SOLUTIONS

12.1 QC

12.1.1 Code: fastp

```
# run fastp like this on the ancestor:
fastp --detect_adapter_for_pe --overrepresentation_analysis --correction  --cut_right --html␣
↪trimmed/anc.fastp.html --json trimmed/anc.fastp.json --thread 2 -i data/anc_R1.fastq.gz -I data/
↪anc_R2.fastq.gz -o trimmed/anc_R1.fastq.gz -O trimmed/anc_R2.fastq.gz

# run the evolved samples through fastp
fastp --detect_adapter_for_pe --overrepresentation_analysis --correction --cut_right --html trimmed/
↪evol1.fastp.html --json trimmed/evol1.fastp.json --thread 2 -i data/evol1_R1.fastq.gz -I data/
↪evol1_R2.fastq.gz -o trimmed/evol1_R1.fastq.gz -O trimmed/evol1_R2.fastq.gz

fastp --detect_adapter_for_pe --overrepresentation_analysis --correction --cut_right --html trimmed/
↪evol2.fastp.html --json trimmed/evol2.fastp.json --thread 2 -i data/evol2_R1.fastq.gz -I data/
↪evol2_R2.fastq.gz -o trimmed/evol2_R1.fastq.gz -O trimmed/evol2_R2.fastq.gz
```

12.1.2 Code: FastQC

Create directory:

```
mkdir trimmed-fastqc
```

Run FastQC:

```
fastqc -o trimmed-fastqc trimmed/*.fastq.gz
```

Run MultiQC

```
multiqc trimmed-fastqc trimmed
```

Open |multiqc| report html webpage:

```
firefox multiqc_report.html
```

12.2 Assembly

12.2.1 Code: SPAdes assembly (trimmed data)

```
spades.py -o assembly/spades-150/ --careful -1 trimmed/anc_R1.fastq.gz -2 trimmed/anc_R2.fastq.gz
```

12.2.2 Code: SPAdes assembly (original data)

```
spades.py -o assembly/spades-original/ --careful -1 data/anc_R1.fastq.gz -2 data/anc_R2.fastq.gz
```

12.2.3 Code: Quast

```
quast -o assembl/quast assembly/spades-150/scaffolds.fasta assembly/spades-original/scaffolds.fasta
```

12.3 Mapping

12.3.1 Code: BWA indexing

Index the genome assembly:

```
bwa index assembly/scaffolds.fasta
```

12.3.2 Code: BWA mapping

Run bwa mem:

```
# trimmed data
bwa mem assembly/scaffolds.fasta trimmed/evol1_R1.fastq.gz trimmed/evol1_R2.fastq.gz > mappings/
↪evol1.sam
bwa mem assembly/scaffolds.fasta trimmed/evol2_R1.fastq.gz trimmed/evol2_R2.fastq.gz > mappings/
↪evol2.sam
```

12.3.3 Code: Mapping post-processing

```
#
# Evol 1
#

# fixmate and compress to bam
samtools sort -n -O sam mappings/evol1.sam | samtools fixmate -m -O bam - mappings/evol1.fixmate.bam
rm mappings/evol1.sam
# sort
samtools sort -O bam -o mappings/evol1.sorted.bam mappings/evol1.fixmate.bam
rm mappings/evol1.fixmate.bam
# mark duplicates
samtools markdup -r -S mappings/evol1.sorted.bam mappings/evol1.sorted.dedup.bam
rm mappings/evol1.sorted.bam
# extract q20 mappers
samtools view -h -b -q 20 mappings/evol1.sorted.dedup.bam > mappings/evol1.sorted.dedup.q20.bam
# extract unmapped
```

(continues on next page)

(continued from previous page)

```
samtools view -b -f 4 mappings/evol1.sorted.dedup.bam > mappings/evol1.sorted.unmapped.bam
rm mappings/evol1.sorted.dedup.bam
# covert to fastq
samtools fastq -1 mappings/evol1.sorted.unmapped.R1.fastq.gz -2 mappings/evol1.sorted.unmapped.R2.
↪fastq.gz mappings/evol1.sorted.unmapped.bam
# delete not needed files
rm mappings/evol1.sorted.unmapped.bam

#
# Evol 2
#

samtools sort -n -O sam mappings/evol2.sam | samtools fixmate -m -O bam - mappings/evol2.fixmate.bam
rm mappings/evol2.sam
samtools sort -O bam -o mappings/evol2.sorted.bam mappings/evol2.fixmate.bam
rm mappings/evol2.fixmate.bam
samtools markdup -r -S mappings/evol2.sorted.bam mappings/evol2.sorted.dedup.bam
rm mappings/evol2.sorted.bam
samtools view -h -b -q 20 mappings/evol2.sorted.dedup.bam > mappings/evol2.sorted.dedup.q20.bam
rm mappings/evol2.sorted.dedup.bam
```

12.3.4 Code: Variant calling

```
# index genome
samtools faidx assembly/scaffolds.fasta
mkdir variants

#
# Evol 1
#

# index mappings
bamtools index -in mappings/evol1.sorted.dedup.q20.bam

# calling variants
freebayes -p 1 -f assembly/scaffolds.fasta mappings/evol1.sorted.dedup.q20.bam > variants/evol1.
↪freebayes.vcf
# compress
bgzip variants/evol1.freebayes.vcf
# index
$ tabix -p vcf variants/evol1.freebayes.vcf.gz

# filtering
zcat variants/evol1.freebayes.vcf.gz | vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR > 0 &
↪ RPR > 1 & RPL > 1" | bgzip > variants/evol1.freebayes.filtered.vcf.gz
tabix -p vcf variants/evol1.freebayes.filtered.vcf.gz

#
# Evol 2
#

# index mappings
bamtools index -in mappings/evol2.sorted.dedup.q20.bam

# calling variants
freebayes -p 1 -f assembly/scaffolds.fasta mappings/evol2.sorted.dedup.q20.bam > variants/evol2.
↪freebayes.vcf
# compress
bgzip variants/evol2.freebayes.vcf
```

(continues on next page)

(continued from previous page)

```
# index
$ tabix -p vcf variants/evol2.freebayes.vcf.gz

# filtering
zcat variants/evol2.freebayes.vcf.gz | vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR > 0 &
↪ RPR > 1 & RPL > 1" | bgzip > variants/evol2.freebayes.filtered.vcf.gz
tabix -p vcf variants/evol2.freebayes.filtered.vcf.gz
```

CHAPTER

THIRTEEN

DOWNLOADS

13.1 Tools

- Miniconda installer [EXTERNAL[225]]
- Minikraken database [EXTERNAL[226]]
- Centrifuge database [EXTERNAL[227]]
- Krona taxonomy database [DROPBOX[228]]

13.2 Data

- *Quality control* (page 9): Raw data-set [DROPBOX[229]] [OSF[230]]
- *Quality control* (page 9): Trimmed data-set [DROPBOX[231]] [OSF[232]]
- *Genome assembly* (page 19): Assembled data-set [DROPBOX[233]] [OSF[234]]
- *Read mapping* (page 25): Mapping index (bwa) [DROPBOX[235]] [OSF[236]]
- *Read mapping* (page 25): Mapped data [DROPBOX[237]] [OSF[238]]
- *Variant calling* (page 47): Called/filtered variants [DROPBOX[239]] [OSF[240]]
- *Genome annotation* (page 55): GFF annotation file [DROPBOX[241]] [OSF[242]]
- *Variants-of-interest* (page 65): SnpEff annotated vcf-file [DROPBOX[243]] [OSF[244]]

[225] https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
[226] ftp://ftp.ccb.jhu.edu/pub/data/kraken2_dbs/minikraken2_v2_8GB_201904_UPDATE.tgz
[227] ftp://ftp.ccb.jhu.edu/pub/infphilo/centrifuge/data/p_compressed+h+v.tar.gz
[228] https://www.dropbox.com/s/cwf1qc5zyq65yvn/taxonomy.tab.gz?dl=0
[229] https://www.dropbox.com/s/3vu1mct230ewhwl/data.tar.gz?dl=0
[230] https://osf.io/2jc4a/download
[231] https://www.dropbox.com/s/y3xsggn0glb6ter/trimmed.tar.gz?dl=0
[232] https://osf.io/m3wpr/download
[233] https://www.dropbox.com/s/h906x9maw879t5s/assembly.tar.gz?dl=0
[234] https://osf.io/t2zpm/download
[235] https://www.dropbox.com/s/ii3vbdj9yn916k4/mapping_idx.tar.gz?dl=0
[236] https://osf.io/tnzrf/download
[237] https://www.dropbox.com/s/8bporren0o230oo/mappings.tar.gz?dl=0
[238] https://osf.io/g5at8/download
[239] https://www.dropbox.com/s/lraiepofsvkl1md/variants.tar.gz?dl=0
[240] https://osf.io/4nzrm/download
[241] https://www.dropbox.com/s/16p9tb22lsvqxbg/annotation.tar.gz?dl=0
[242] https://osf.io/7t4yh/download
[243] https://www.dropbox.com/s/yzbu0eealf7xfr1/voi.tar.gz?dl=0
[244] https://osf.io/5c6w9/download

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[KAWECKI2012] Kawecki TJ et al. Experimental evolution. Trends in Ecology and Evolution (2012) 27:10[5]

[ZEYL2006] Zeyl C. Experimental evolution with yeast. FEMS Yeast Res, 2006, 685–691[6]

[GLENN2011] Glenn T. Field guide to next-generation DNA sequencers. Molecular Ecology Resources (2011) 11, 759–769 doi: 10.1111/j.1755-0998.2011.03024.x[41]

[KIRCHNER2014] Kirchner et al. Addressing challenges in the production and analysis of Illumina sequencing data. BMC Genomics (2011) 12:382[42]

[MUKHERJEE2015] Mukherjee S, Huntemann M, Ivanova N, Kyrpides NC and Pati A. Large-scale contamination of microbial isolate genomes by Illumina PhiX control. Standards in Genomic Sciences, 2015, 10:18. DOI: 10.1186/1944-3277-10-18[43]

[ROBASKY2014] Robasky et al. The role of replicates for error mitigation in next-generation sequencing. Nature Reviews Genetics (2014) 15, 56-62[44]

[ABBAS2014] Abbas MM, Malluhi QM, Balakrishnan P. Assessment of de novo assemblers for draft genomes: a case study with fungal genomes. BMC Genomics. 2014;15 Suppl 9:S10.[64] doi: 10.1186/1471-2164-15-S9-S10. Epub 2014 Dec 8.

[COMPEAU2011] Compeau PE, Pevzner PA, Tesler G. How to apply de Bruijn graphs to genome assembly. Nat Biotechnol. 2011 Nov 8;29(11):987-91[65]

[GUREVICH2013] Gurevich A, Saveliev V, Vyahhi N and Tesler G. QUAST: quality assessment tool for genome assemblies. Bioinformatics 2013, 29(8), 1072-1075[66]

[NAGARAJAN2013] Nagarajan N, Pop M. Sequence assembly demystified. Nat Rev Genet. 2013 Mar;14(3):157-67[67]

[SALZBERG2012] Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, Treangen TJ, Schatz MC, Delcher AL, Roberts M, Marçais G, Pop M, Yorke JA. GAGE: A critical evaluation of genome assemblies and assembly algorithms. Genome Res. 2012 Mar;22(3):557-67[68]

[WICK2015] Wick RR, Schultz MB, Zobel J and Holt KE. Bandage: interactive visualization of de novo genome assemblies. Bioinformatics 2015, 10.1093/bioinformatics/btv383[69]

[5] http://dx.doi.org/10.1016/j.tree.2012.06.001
[6] http://doi.org/10.1111/j.1567-1364.2006.00061.x
[41] http://doi.org/10.1111/j.1755-0998.2011.03024.x
[42] http://doi.org/10.1186/1471-2164-12-382
[43] http://doi.org/10.1186/1944-3277-10-18
[44] http://doi.org/10.1038/nrg3655
[64] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4290589/
[65] http://dx.doi.org/10.1038/nbt.2023
[66] http://bioinformatics.oxfordjournals.org/content/29/8/1072
[67] http://dx.doi.org/10.1038/nrg3367
[68] http://genome.cshlp.org/content/22/3/557.full?sid=59ea80f7-b408-4a38-9888-3737bc670876
[69] http://bioinformatics.oxfordjournals.org/content/early/2015/07/11/bioinformatics.btv383.long

[LI2009] Li H, Durbin R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics. 25 (14): 1754–1760.[103]

[OKO2015] Okonechnikov K, Conesa A, García-Alcalde F. Qualimap 2: advanced multi-sample quality control for high-throughput sequencing data. Bioinformatics (2015), 32, 2:292–294.[104]

[KIM2017] Kim D, Song L, Breitwieser FP, Salzberg SL. Centrifuge: rapid and sensitive classification of metagenomic sequences. Genome Res. 2016 Dec;26(12):1721-1729[167]

[LU2017] Lu J, Breitwieser FP, Thielen P, Salzberg SL. Bracken: estimating species abundance in metagenomics data. PeerJ Computer Science, 2017, 3:e104, doi:10.7717/peerj-cs.104[168]

[ONDOV2011] Ondov BD, Bergman NH, and Phillippy AM. Interactive metagenomic visualization in a Web browser. BMC Bioinformatics, 2011, 12(1):385.[169]

[WOOD2014] Wood DE and Steven L Salzberg SL. Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology, 2014, 15:R46. DOI: 10.1186/gb-2014-15-3-r46[170].

[NIELSEN2011] Nielsen R, Paul JS, Albrechtsen A, Song YS. Genotype and SNP calling from next-generation sequencing data. Nat Rev Genetics, 2011, 12:433-451[180]

[OLSEN2015] Olsen ND et al. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. Front. Genet., 2015, 6:235.[181]

[SIMAO2015] Simao FA, Waterhouse RM, Ioannidis P, Kriventseva EV and Zdobnov EM. BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. Bioinformatics, 2015, Oct 1;31(19):3210-2[200]

[STANKE2005] Stanke M and Morgenstern B. AUGUSTUS: a web server for gene prediction in eukaryotes that allows user-defined constraints. Nucleic Acids Res, 2005, 33(Web Server issue): W465–W467.[201]

[103] https://doi.org/10.1093%2Fbioinformatics%2Fbtp324
[104] https://doi.org/10.1093/bioinformatics/btv566
[167] https://www.ncbi.nlm.nih.gov/pubmed/27852649
[168] https://peerj.com/articles/cs-104/
[169] http://www.ncbi.nlm.nih.gov/pubmed/21961884
[170] http://doi.org/10.1186/gb-2014-15-3-r46
[180] http://doi.org/10.1038/nrg2986
[181] https://doi.org/10.3389/fgene.2015.00235
[200] http://doi.org/10.1093/bioinformatics/btv351
[201] https://dx.doi.org/10.1093/nar/gki458