Computational Genomics Tutorial

Release 2018.04

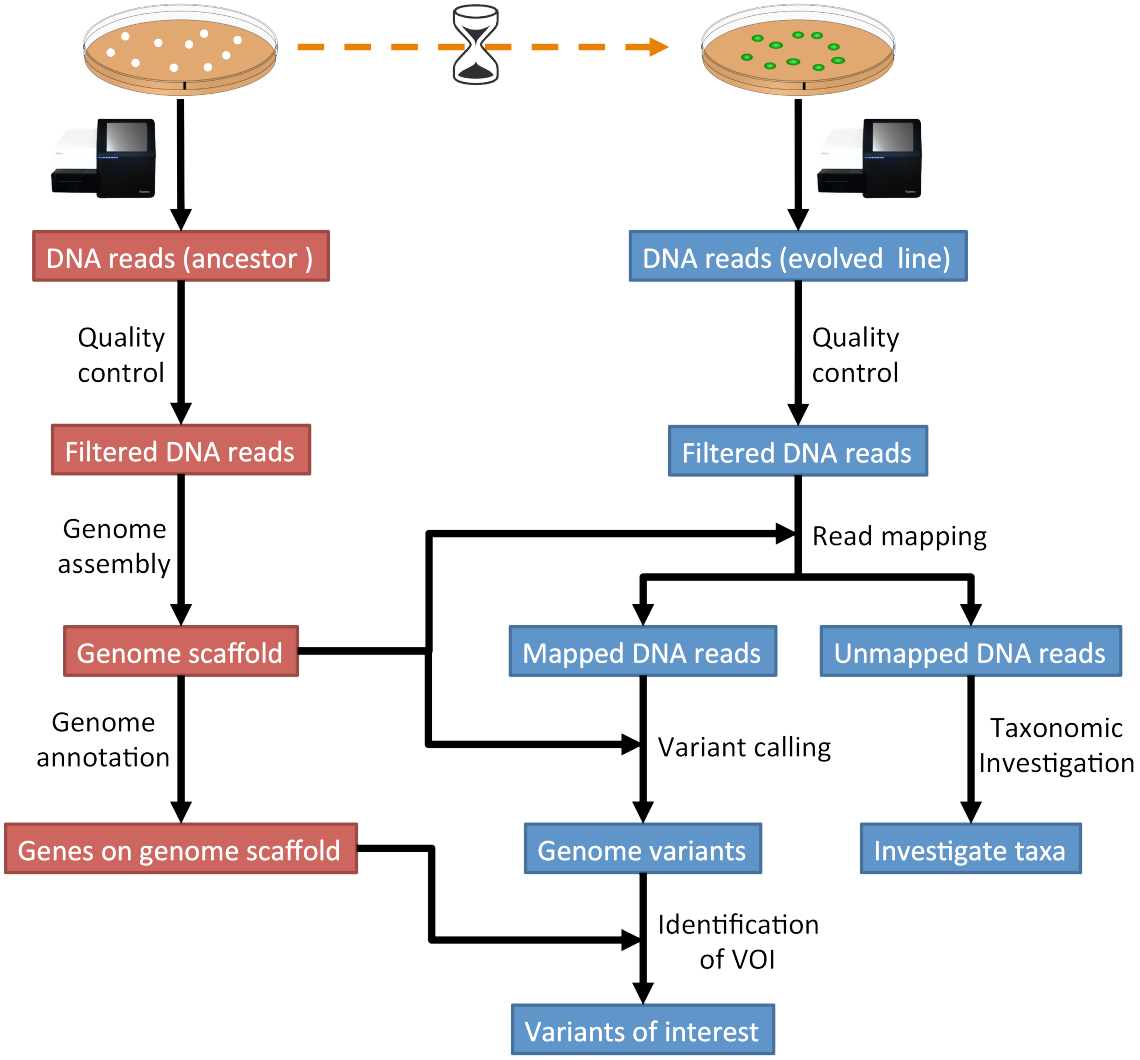
Sebastian Schmeier (https://sschmeier.com)

Apr 20, 2018

CONTENTS

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. Currently Sebastian[1] is teaching this material in the Massey University course Genetics and Evolution[2].

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



[1] https://sschmeier.com
[2] https://www.massey.ac.nz/massey/learning/programme-course/course.cfm?paper_code=203.341
[3] https://sschmeier.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]* (page 85), *[ZEYL2006]* (page 85). The final aim is to identify the genome variations in evolved lines of wild yeast that can explain the observed biological phenotype.

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

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

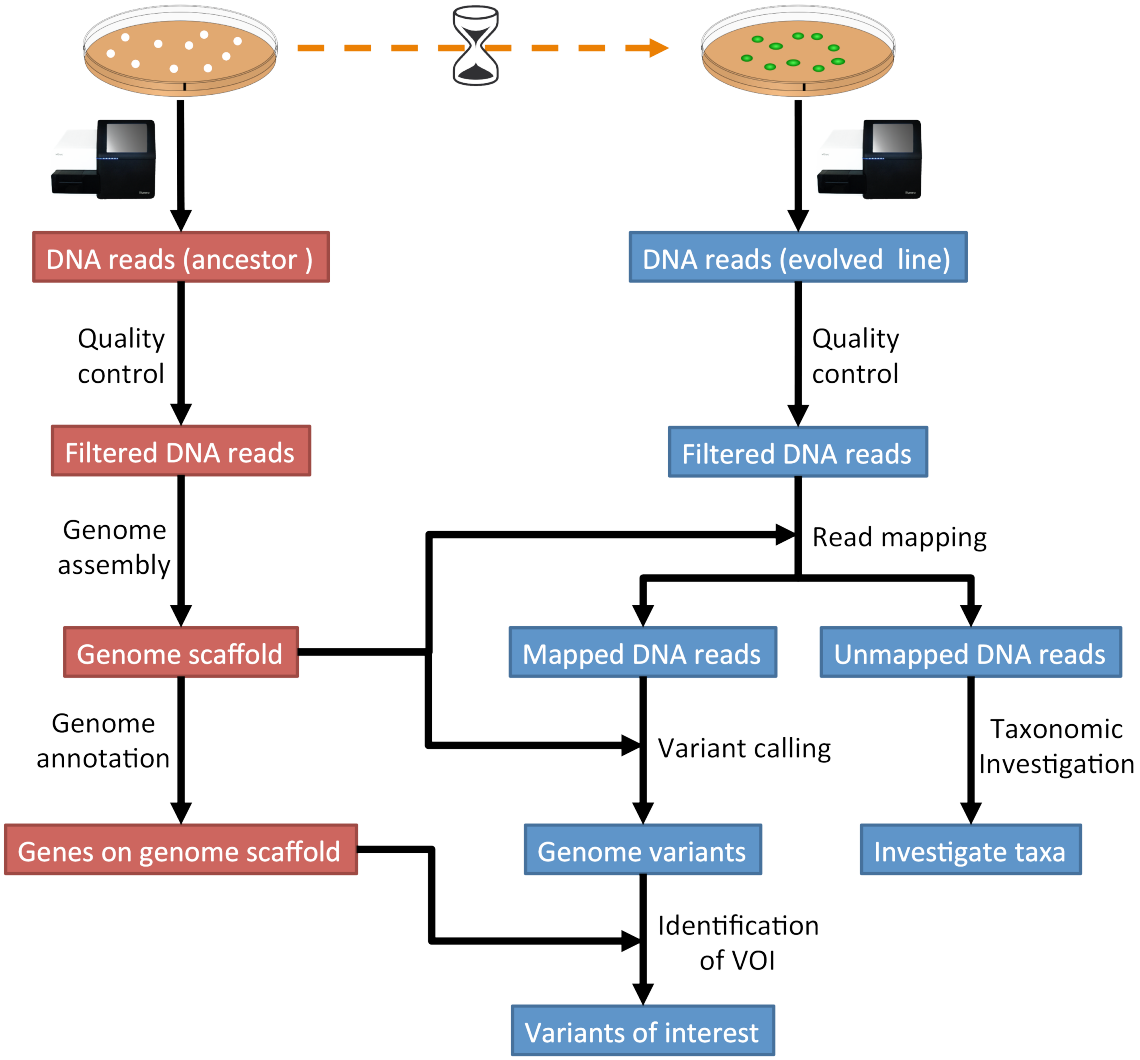


Fig. 1.1: The tutorial will follow this workflow.

CHAPTER

TWO

TOOL INSTALLATION

2.1 Install the conda package manager

We will use the package/tool managing system conda[34] 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.

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

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

# delete the installer after successful run
rm Miniconda3-latest-Linux-x86_64.sh
```

Note: Should the conda installer download fail. Please find links to alternative locations on the *Downloads* (page 79) page.

2.1.1 Update `.bashrc` and `.zshrc` config-files

Before we are able to use conda[35] we need to tell our shell where it can find the program. We add the right path to the conda[36] installation to our shell config files:

```
echo 'export PATH="/home/manager/miniconda3/bin:$PATH"' >> ~/.bashrc
echo 'export PATH="/home/manager/miniconda3/bin:$PATH"' >> ~/.zshrc
```

Attention: The above assumes that your username is "manager", which is the default on a Biolinux install. Replace "manager" with your actual username. Find out with `whoami`.

So what is actually happening here? We are appending a line to a file (either `.bashrc` or `.zshrc`). If we are starting a new command-line shell, either file gets executed first (depending on which shell you are using, either bash or zsh shells). What this line does, is to put permanently the directory `~/miniconda3/bin` first on your `PATH` variable. The `PATH` variable contains directories in which our computer looks for installed programs, one directory after the other until the program you requested is found (or not, then it will complain). Through the addition of the above line we make sure that the program `conda` can be found anytime we open a new shell.

Close shell/terminal, **re-open** new shell/terminal. Now, we should be able to use the conda[37] command:

[34] http://conda.pydata.org/miniconda.html
[35] http://conda.pydata.org/miniconda.html
[36] http://conda.pydata.org/miniconda.html
[37] http://conda.pydata.org/miniconda.html

```
conda update conda
```

2.1.2 Installing conda channels to make tools available

Different tools are packaged in what conda[38] 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 conda-forge
conda config --add channels bioconda
```

2.2 Create environments

We create a conda[39] 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 (mentioned above) gets temporarily manipulated and set to:

```
$ conda activate ngs
# Lets look at the content of the PATH variable
(ngs) $ echo $PATH
/home/manager/miniconda3/envs/ngs/bin:/home manager/miniconda3/bin:/usr/local/bin: ...
```

Now it will look first in your environment's bin directory but afterwards in the general conda bin (/home/manager/miniconda3/bin). 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/manager/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.

[38] http://conda.pydata.org/miniconda.html
[39] 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]

# deavtivate 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]* (page 85). 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]* (page 85).

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
curl -O http://compbio.massey.ac.nz/data/203341/data.tar.gz

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

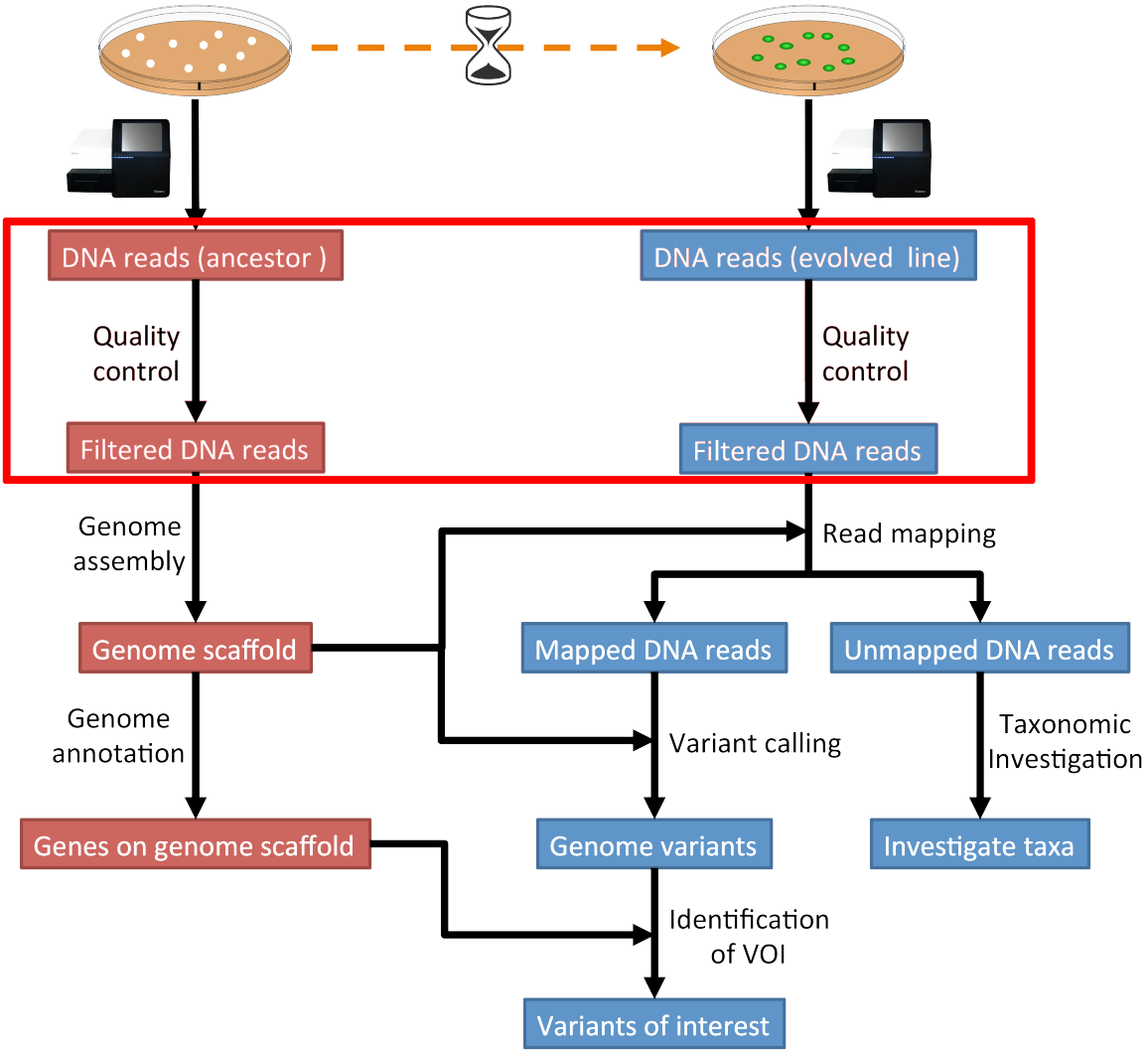


Fig. 3.1: The part of the workflow we will work on in this section marked in red.

Note: Should the download fail, download manually from *Downloads* (page 79).

The data is from a paired-end sequencing run data (see Fig. 3.2) from an Illumina[67] MiSeq *[GLENN2011]* (page 85). Thus, we have two files, one for each end of the read.

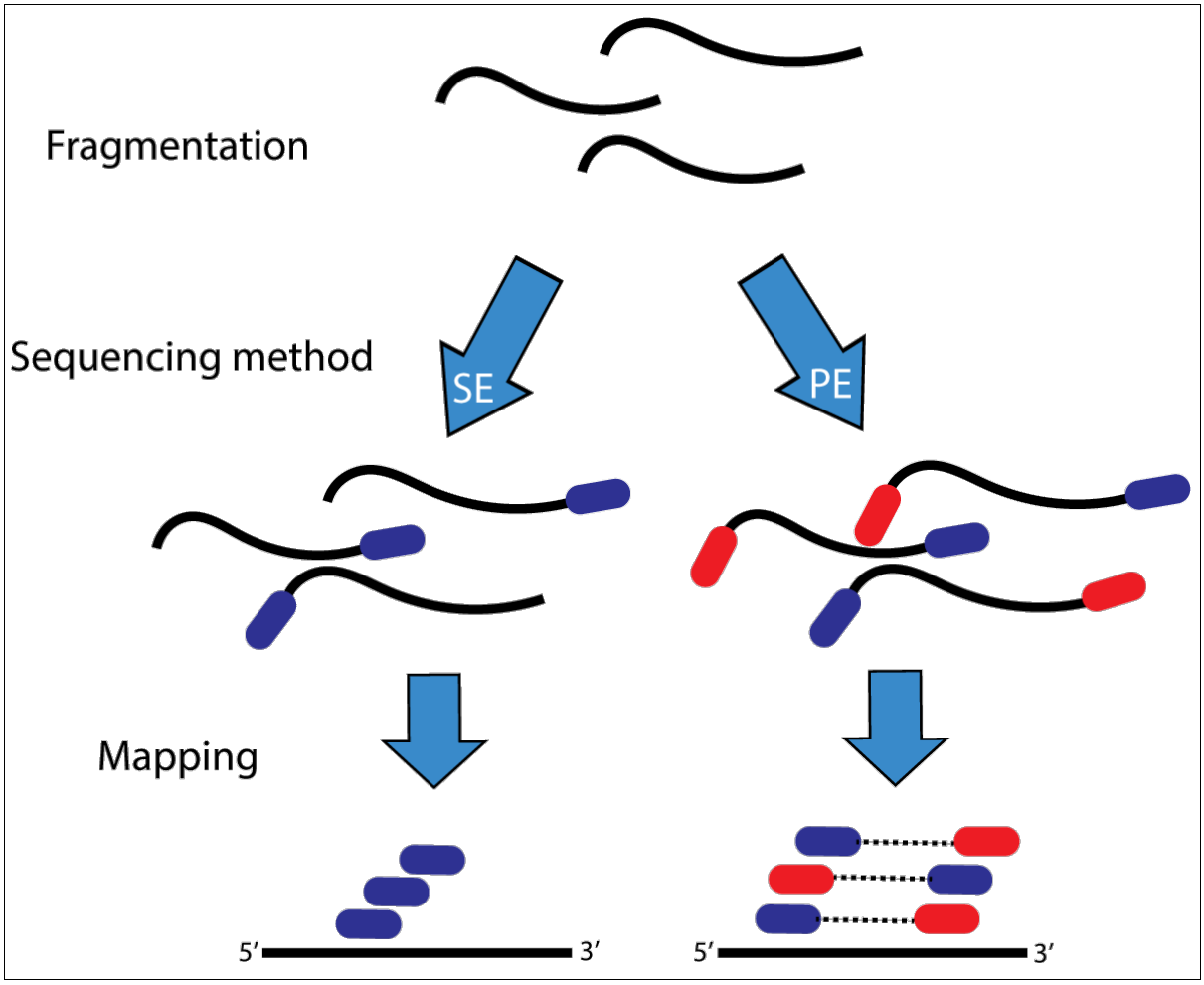


Fig. 3.2: Illustration of single-end (SE) versus paired-end (PE) sequencing.

If you need to refresh how Illumina[68] paired-end sequencing works have a look at the Illumina technology webpage[69] and this video[70].

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[71] adapters have been removed as well already! The process is explained below but we are **not** going to do it.

[67] http://illumina.com
[68] http://illumina.com
[69] http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html
[70] https://youtu.be/HMyCqWhwB8E
[71] 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 12MB. 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. 2x100 paired-end = 200)
- `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[72]!

A useful tool to decode base qualities can be found here[73].

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
2. Adapter trimming
3. Quality trimming of reads
4. Quality assessment

3.7 PhiX genome

PhiX[74] 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[75]. 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]* (page 85). The steps involve mapping all reads to the "known" PhiX genome, and removing all of those sequence reads from the data.

[72] https://en.wikipedia.org/wiki/FASTQ_format
[73] http://broadinstitute.github.io/picard/explain-qualities.html
[74] https://en.wikipedia.org/wiki/Phi_X_174
[75] 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 this has been already done. Please see the *Read mapping* (page 29) section on how to map reads against a reference genome.

3.8 Adapter trimming

The process of sequencing DNA via Illumina[76] 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.

Attention: The process of how to do this is explained here, however we are **not** going to do this as our sequences have been adapter-trimmed already.

First, 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[77] itself provides a document[78] that describes the adapters used for their different technologies. Also the FastQC[79] tool, we will be using later on, provides a collection of contaminants and adapters[80].

Second, we need a tool that takes a list of adapters and scans each sequence read and removes the adapters. Install a tool called fastq-mcf[81] from the ea-utils suite[82] of tools that is able to do this.

```
# install
conda install ea-utils
```

Using the tool together with a adapter/contaminants list in fasta-file (here denoted as `adapters.fa`):

```
fastq-mcf -o cleaned.R1.fq.gz -o cleaned.R2.fq.gz adapaters.fa infile_R1.fastq infile_R2.fastq
```

- `-o`: Specifies the output-files. These are fastq-files for forward and reverse read, with adapters removed.

3.9 Quality assessment of sequencing reads (SolexaQA++)

To assess the sequence read quality of the Illumina[83] run we make use of a program called SolexaQA++[84] *[COX2010]* (page 85). SolexaQA++[85] was originally developed to work with Solexa data (since bought by Illumina[86]), but long since working with Illumina[87] data. It produces nice graphics that intuitively show the quality of the sequences. it is also able to dynamically trim the bad quality ends off the reads.

From the webpage:

[76] http://illumina.com
[77] http://illumina.com
[78] https://support.illumina.com/downloads/illumina-customer-sequence-letter.html
[79] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[80] https://github.com/csf-ngs/fastqc/blob/master/Contaminants/contaminant_list.txt
[81] https://github.com/ExpressionAnalysis/ea-utils/blob/wiki/FastqMcf.md
[82] https://expressionanalysis.github.io/ea-utils/
[83] http://illumina.com
[84] http://solexaqa.sourceforge.net
[85] http://solexaqa.sourceforge.net
[86] http://illumina.com
[87] http://illumina.com

> "SolexaQA calculates sequence quality statistics and creates visual representations of data quality for second-generation sequencing data. Originally developed for the Illumina system (historically known as "Solexa"), SolexaQA now also supports Ion Torrent and 454 data."

3.9.1 Install SolexaQA++

Unfortunately, currently we cannot install SolexaQA++[88] with conda[89].

```
curl -O http://compbio.massey.ac.nz/data/203341/SolexaQA.tar.gz

# uncompress the archive
tar -xvzf SolexaQA.tar.gz

# make the file executable
chmod a+x SolexaQA/Linux_x64/SolexaQA++

# copy program to root folder
cp ./SolexaQA/Linux_x64/SolexaQA++ .

# run the program
./SolexaQA++
```

Note: Should the download fail, download manually from *Downloads* (page 79).

3.9.2 SolexaQA++ manual

SolexaQA++[90] has three modes that can be run. Type:

```
./SolexaQA++
```

```
SolexaQA++ v3.1.3
Released under GNU General Public License version 3
C++ version developed by Mauro Truglio (M.Truglio@massey.ac.nz)

Usage: SolexaQA++ <command> [options]

Command: analysis      quality analysis and graphs generation
         dynamictrim    trim reads using a chosen threshold
         lengthsort  sort reads by a chosen length
```

The three modes are: `analysis`, `dynamictrim`, and `lengthsort`:

`analysis` - the primary quality analysis and visualization tool. Designed to run on unmodified FASTQ files obtained directly from Illumina[91], Ion Torrent or 454 sequencers.

`dynamictrim` - a read trimmer that individually crops each read to its longest contiguous segment for which quality scores are greater than a user-supplied quality cutoff.

`lengthsort` - a program to separate high quality reads from low quality reads. LengthSort assigns trimmed reads to paired-end, singleton and discard files based on a user-defined length cutoff.

[88] http://solexaqa.sourceforge.net
[89] http://conda.pydata.org/miniconda.html
[90] http://solexaqa.sourceforge.net
[91] http://illumina.com

3.9.3 SolexaQA++ dynamic trimming

We will use SolexaQA++[92] dynamic trim the reads, to chop of nucleotides witha a bad quality score.

Todo:

1. Create a directory for the result-files –> **trimmed/**.
2. Run SolexaQA++[93] `dynamictrim` with the untrimmed data and a probability cutoff of 0.05., and submit result-directory **trimmed/**.
3. Investigate the result-files in **trimmed/**, e.g. do the file-sizes change to the original files?
4. SolexaQA++[94] `dynamictrim` produces a graphical output. Explain what the graph shows. Find heklp on the SolexaQA++[95] website.

Hint: Should you not get 1 and/or 2 right, try the commands in *Code: SolexaQA++ trimming* (page 77).

3.9.4 SolexaQA++ analysis on trimmed data

Todo:

1. Create a directory for the result-files –> **trimmed-solexaqa**.
2. Use SolexaQA++[96] to do the quality assessment with the trimmed data-set.
3. Compare your results to the examples of a particularly bad MiSeq run (Fig. 3.6 to Fig. 3.6, taken from SolexaQA++[97] website). Write down your observations.
4. What elements in these example figures (Fig. 3.3 to Fig. 3.6) indicate that the show a bad run? Write down your explanations.

Hint: Should you not get 1 and/or 2 it right, try the commands in *Code: SolexaQA++ qc* (page 77).

3.10 Sickle for dynamic trimming (alternative to SolexaQA++)

Should the dynamic trimming not work with SolexaQA++[98], you can alternatively use Sickle[99].

```
conda activate ngs
conda install sickle-trim
```

Now we are going to run the program on our paired-end data:

[92] http://solexaqa.sourceforge.net
[93] http://solexaqa.sourceforge.net
[94] http://solexaqa.sourceforge.net
[95] http://solexaqa.sourceforge.net
[96] http://solexaqa.sourceforge.net
[97] http://solexaqa.sourceforge.net
[98] http://solexaqa.sourceforge.net
[99] https://github.com/najoshi/sickle

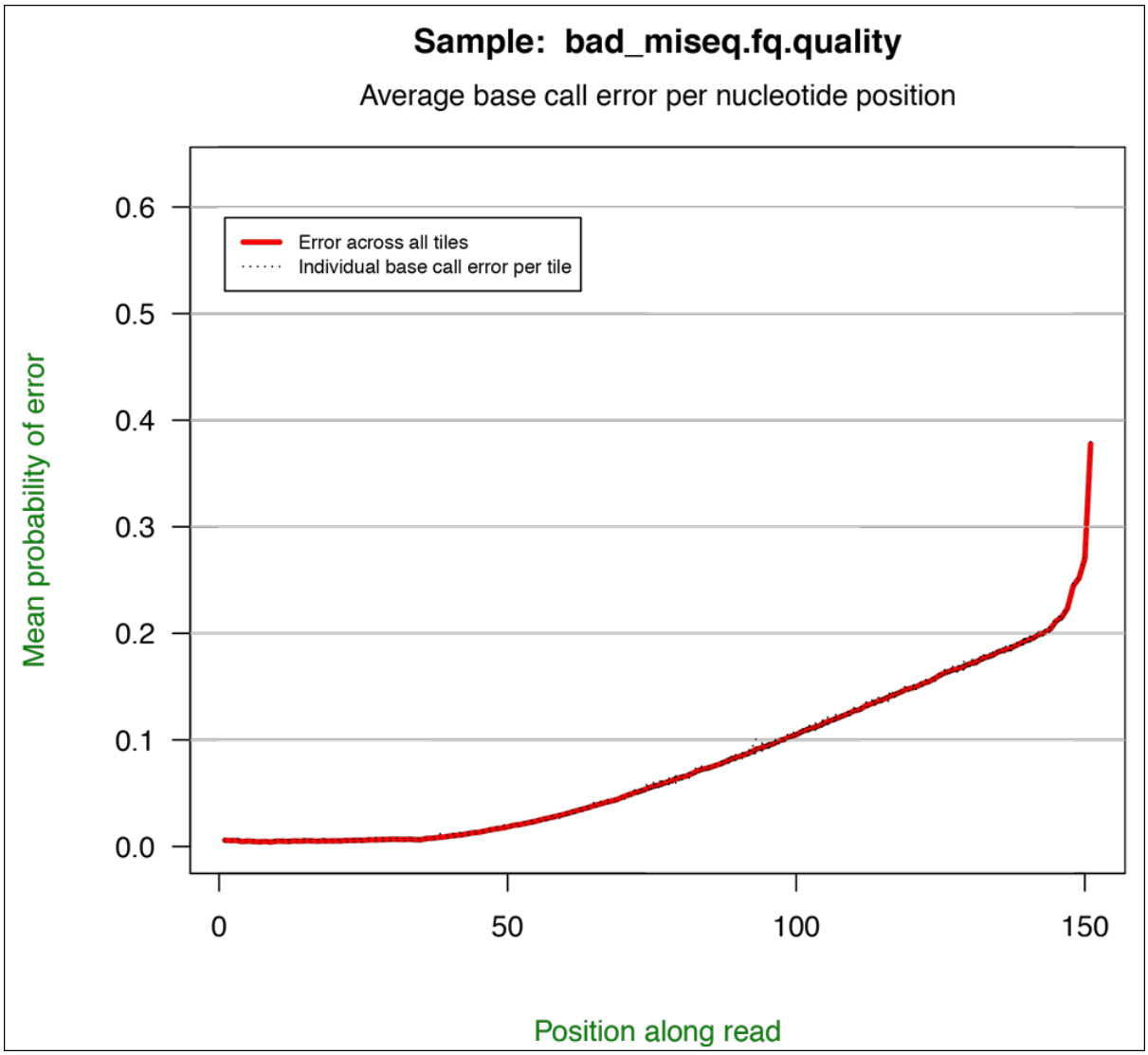


Fig. 3.3: SolexaQA++ example quality plot along reads of a bad MiSeq run

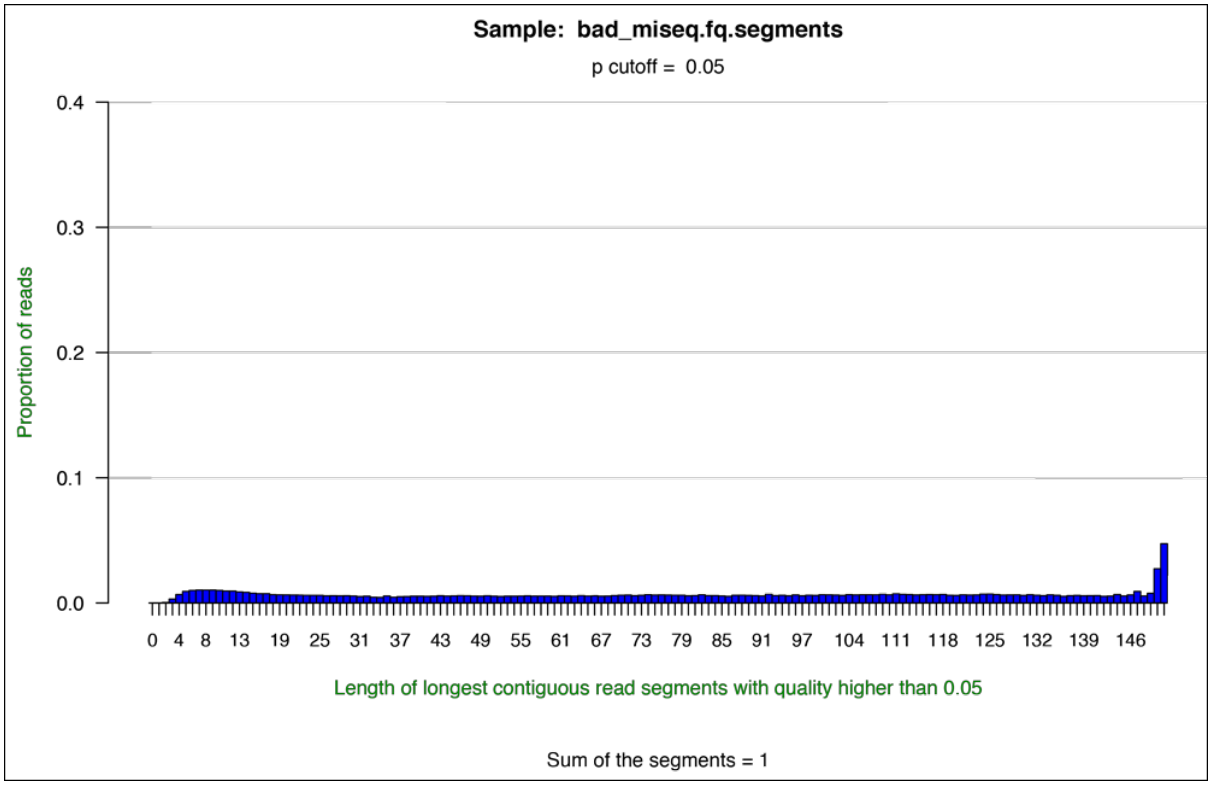


Fig. 3.4: SolexaQA++ example histogram plot of a bad MiSeq run.

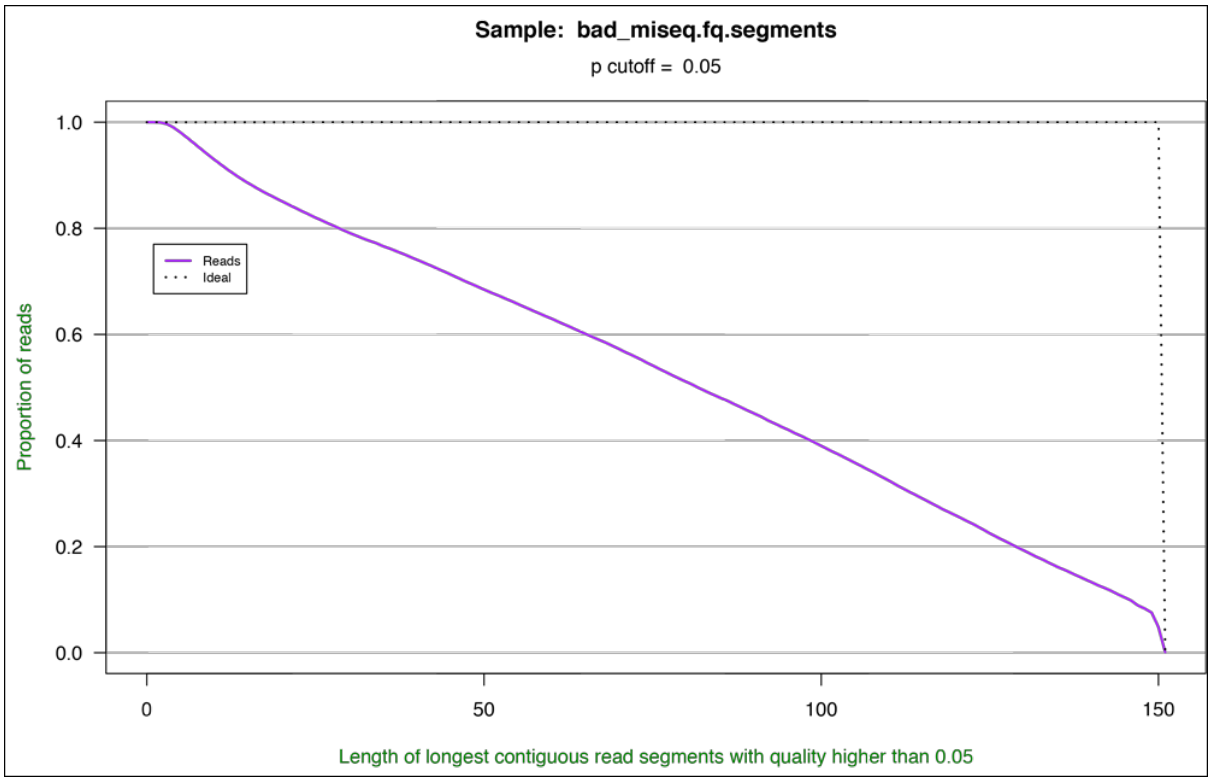


Fig. 3.5: SolexaQA++ example cumulative plot of a bad MiSeq run.

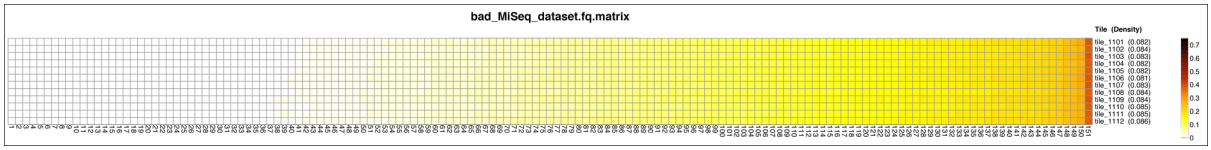


Fig. 3.6: SolexaQA++ example quality heatmap of a bad MiSeq run.

```
# create a new directory
mkdir trimmed

# sickle parameters:
sickle --help

# as we are dealing with paired-end data you will be using "sickle pe"
sickle pe --help

# run sickle like so:
sickle pe -g -t sanger -f data/ancestor-R1.fastq.gz -r data/ancestor-R2.fastq.gz -o trimmed/
↪ancestor-R1.trimmed.fastq.gz -p trimmed/ancestor-R2.trimmed.fastq.gz
```

Hint: Should you be unable to run Sickle[100] or SolexaQA++[101] at all to trim the data. You can download the trimmed dataset here[102]. Unarchive and uncompress the files with `tar -xvzf trimmed.tar.gz`.

3.11 Quality assessment of sequencing reads (FastQC)

3.11.1 Installing FastQC

```
conda activate ngs
conda install fastqc

# should now run the program
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
```

(continues on next page)

[100] https://github.com/najoshi/sickle
[101] http://solexaqa.sourceforge.net
[102] http://compbio.massey.ac.nz/data/203341/trimmed.tar.gz

(continued from previous page)

```
    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.
```

3.11.2 FastQC manual

FastQC[103] is a very simple program to run that provides similar and additional information to SolexaQA++[104].

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.[txt/fa/fq] ...
```

- `-o RESULT-DIR` is the directory where the result files will be written
- `INPUT-FILE.[txt/fa/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[105] made some nice help pages explaining each of the plots and results you expect to see here[106].

3.11.3 Run FastQC on the untrimmed and trimmed data

Todo:

1. Create a directory for the results –> **trimmed-fastqc**
2. Run FastQC on all **trimmed** files.
3. Visit the FastQC[107] website and read about sequencing QC reports for good and bad Illumina[108] sequencing runs.
4. Compare your results to these examples (Fig. 3.7 to Fig. 3.9) of a particularly bad run (taken from the FastQC[109] website) and write down your observations with regards to your data.
5. What elements in these example figures (Fig. 3.7 to Fig. 3.9) indicate that the example is from a bad run?

Hint: Should you not get it right, try the commands in *Code: FastQC* (page 77).

---

[103] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[104] http://solexaqa.sourceforge.net
[105] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[106] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/
[107] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[108] http://illumina.com
[109] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

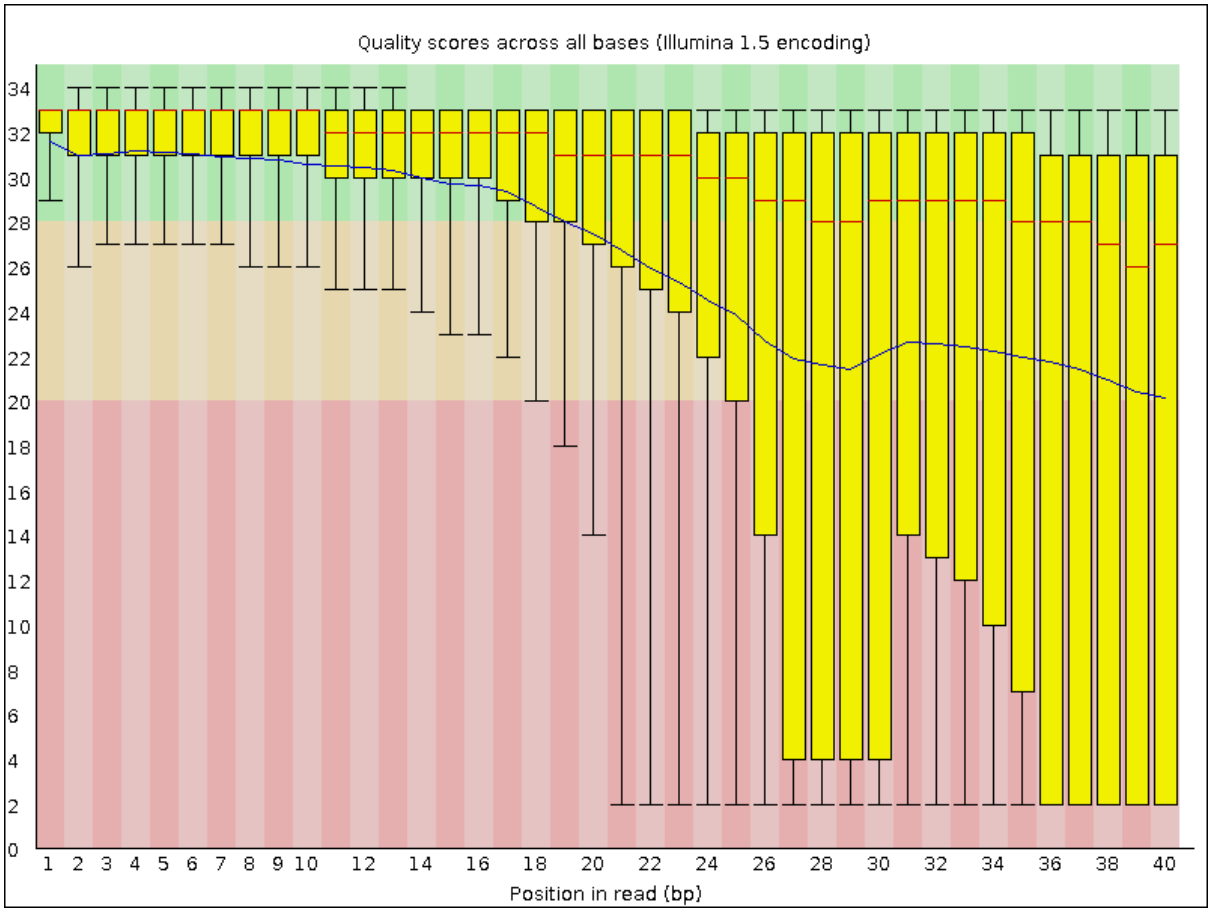


Fig. 3.7: Quality score across bases.

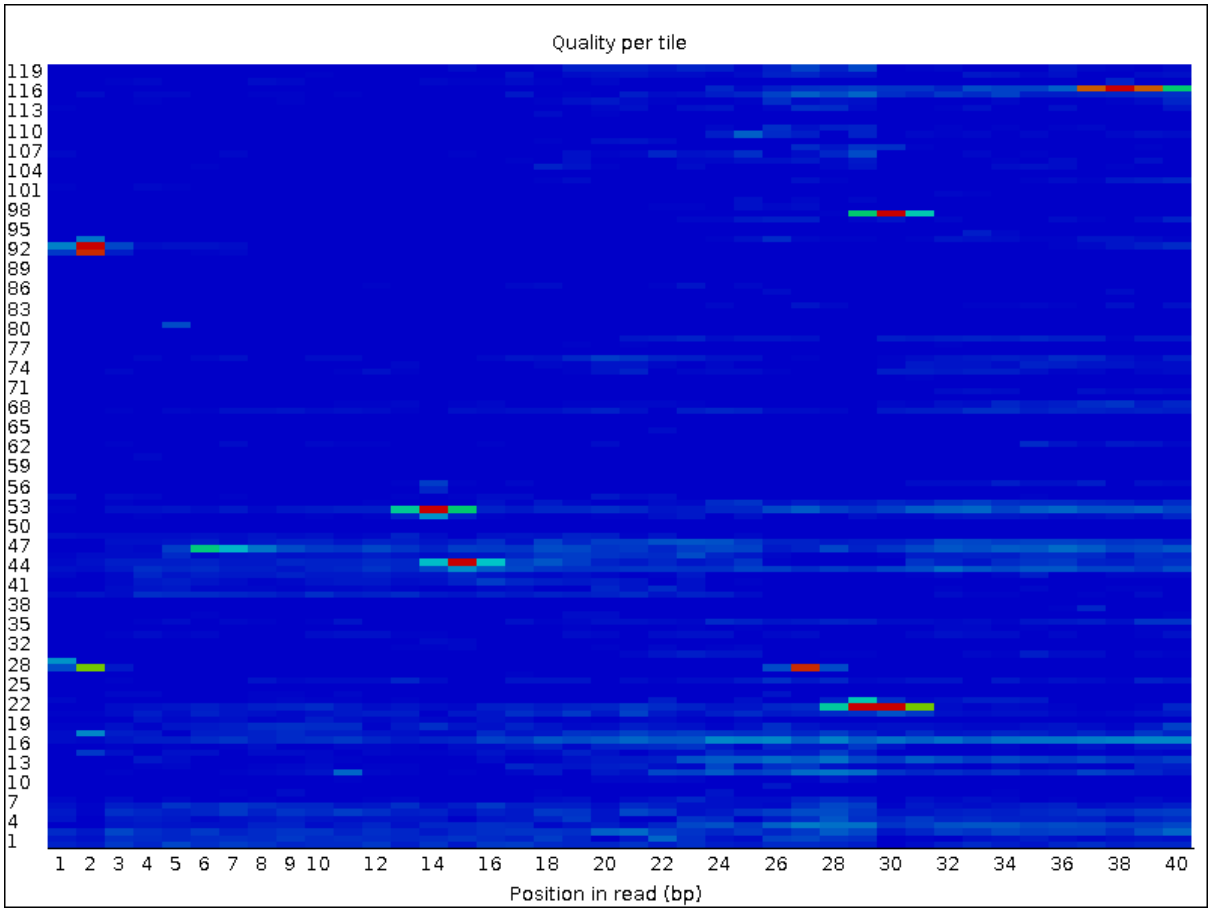


Fig. 3.8: Quality per tile.

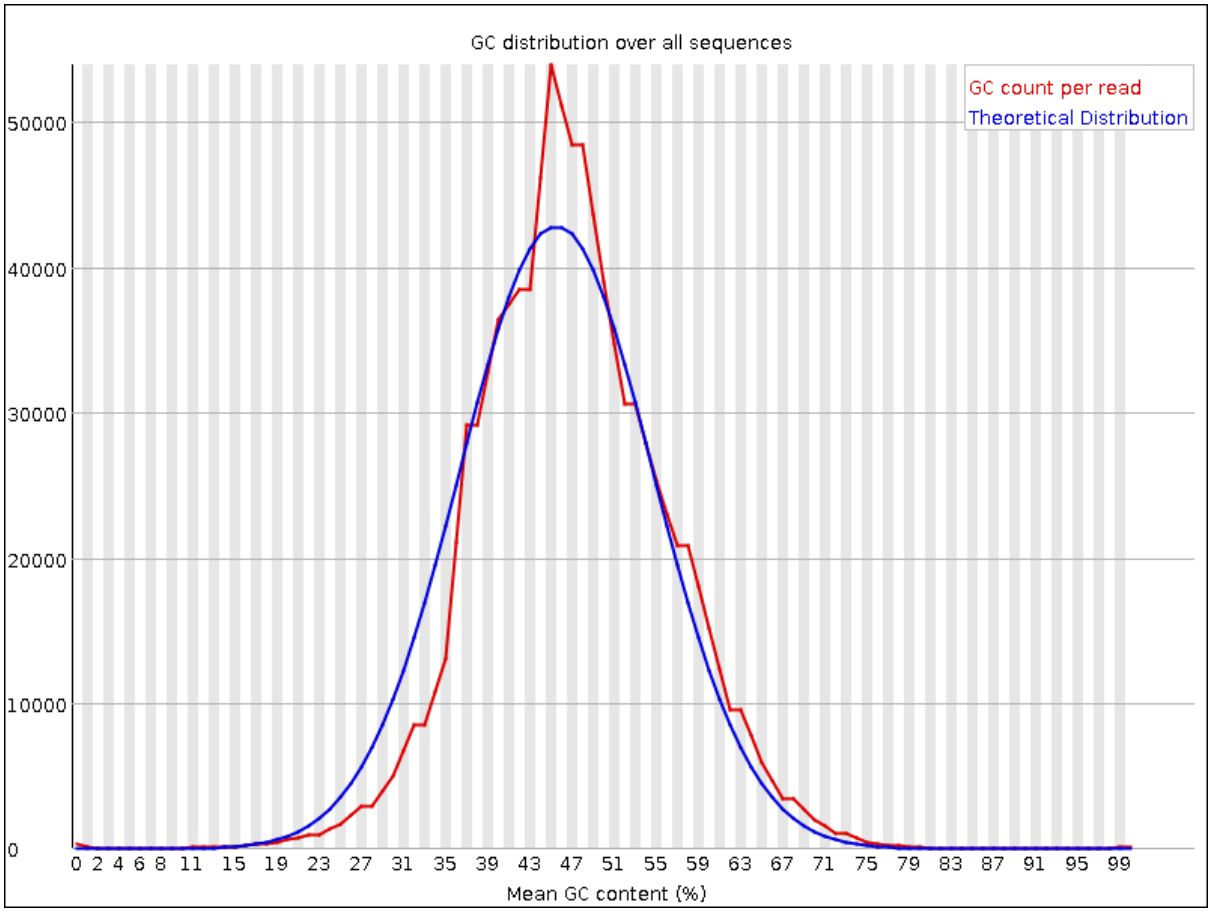


Fig. 3.9: GC distribution over all sequences.

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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
```

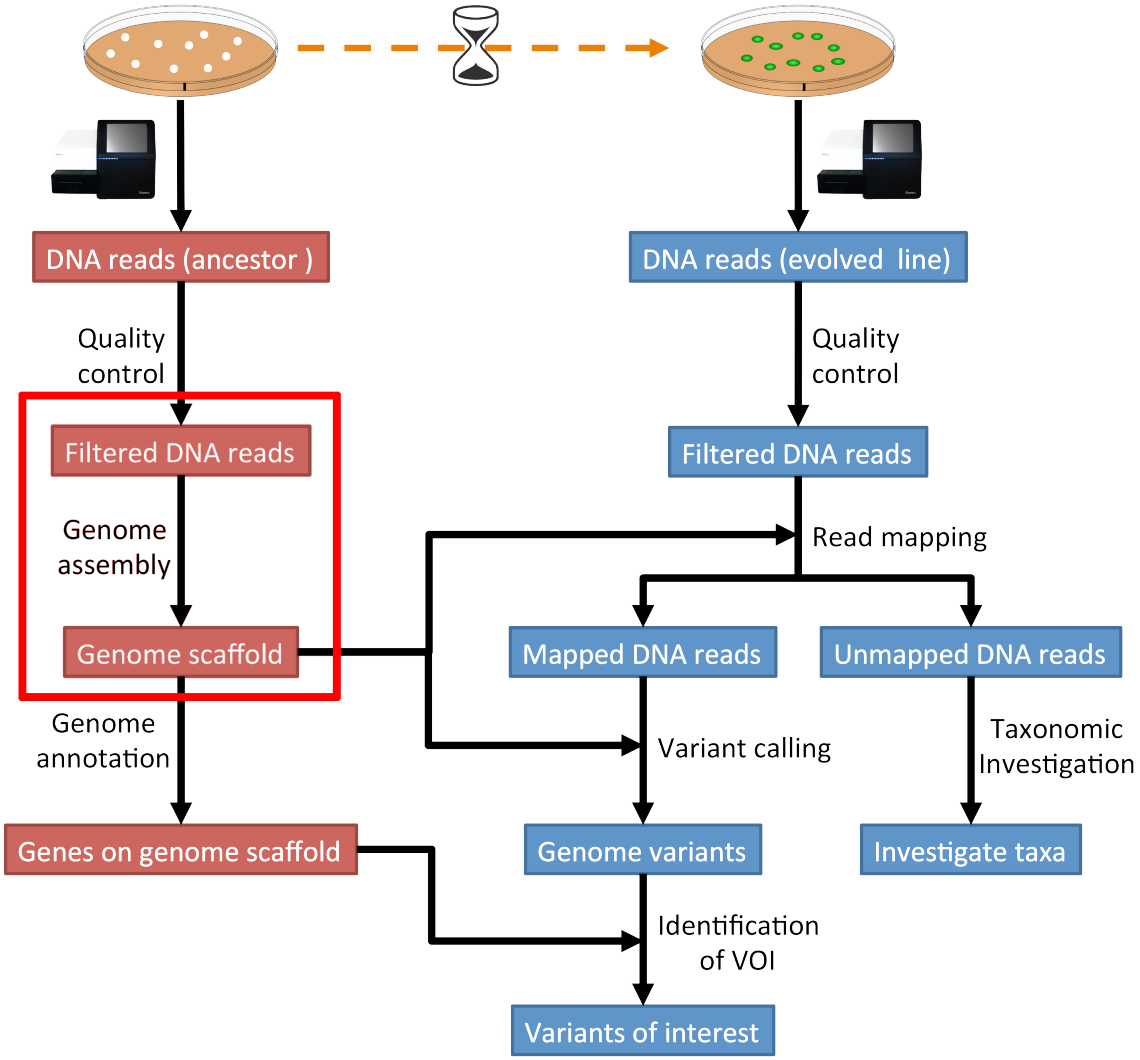


Fig. 4.1: The part of the workflow we will work on in this section marked in red.

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[142] 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[143] fo assembling our genome. In a recent evaluation of assembly software, SPAdes[144] was found to be a good choice for fungal genomes *[ABBAS2014]* (page 85). It is also simple to install and use.

```
conda activate ngs
conda install spades
```

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
```

The two files we need to submit to SPAdes[145] are two paired-end read files.

```
spades.py -o assembly/spades-default/ -1 trimmed/ancestor-R1.fastq.trimmed.gz -2 trimmed/ancestor-
↪R2.fastq.trimmed.gz
```

Todo:

1. Run SPAdes[146] with default parameters on the ancestor
2. Read in the SPAdes[147] manual about about assembling with 2x150bp reads
3. Run SPAdes[148] a second time but use the options suggested at the SPAdes[149] manual section 3.4[150] for assembling 2x150bp paired-end reads (are fungi multicellular?). Use a different output directory `assembly/spades-150` for this run.

---

[142] http://bioinf.spbau.ru/spades
[143] http://bioinf.spbau.ru/spades
[144] http://bioinf.spbau.ru/spades
[145] http://bioinf.spbau.ru/spades
[146] http://bioinf.spbau.ru/spades
[147] http://bioinf.spbau.ru/spades
[148] http://bioinf.spbau.ru/spades
[149] http://bioinf.spbau.ru/spades
[150] http://spades.bioinf.spbau.ru/release3.9.1/manual.html#sec3.4

Hint: Should you not get it right, try the commands in *Code: SPAdes assembly (trimmed data)* (page 78).

4.6 Assembly quality assessment

4.6.1 Assembly statistics

Quast[151] (QUality ASsesment Tool) *[GUREVICH2013]* (page 85), 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
- missassemblies: misassembled and unaligned contigs or contigs bases
- genes and operons covered

It is easy with Quast[152] to compare these measures among several assemblies. The program can be used on their website[153].

As of March 2018, Quast[154] needs Python 2 instead of the default Python 3 we have been using so far. Thus, we make a new conda[155] environment for Quast[156] based on Python 2.

```
conda create -n quast python=2 quast
conda activate quast
```

Run Quast[157] with both assembly scaffolds.fasta files to compare the results.

Note: Should you be unable to run SPAdes[158] on the data, you can manually download the assembly from *Downloads* (page 79). Unarchive and uncompress the files with `tar -xvzf assembly.tar.gz`.

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

Todo:

1. Compare the results of Quast[159] with regards to the two different assemblies.
2. Which one do you prefer and why?

4.7 Compare the untrimmed data

Todo:

[151] http://quast.bioinf.spbau.ru/
[152] http://quast.bioinf.spbau.ru/
[153] http://quast.bioinf.spbau.ru/
[154] http://quast.bioinf.spbau.ru/
[155] http://conda.pydata.org/miniconda.html
[156] http://quast.bioinf.spbau.ru/
[157] http://quast.bioinf.spbau.ru/
[158] http://bioinf.spbau.ru/spades
[159] http://quast.bioinf.spbau.ru/

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[160] on the assembly and compare the statistics to the one derived for the trimmed data set. Write down your observations.

Hint: Should you not get it right, try the commands in *Code: SPAdes assembly (original data)* (page 78).

4.8 Assemblathon

Todo: Now that you know the basics for assembling a genome and judging their quality, play with the SPAdes[161] parameters and the **trimmed data** to create the best assembly possible. We will compare the assemblies to find out who created the best one.

Todo:

1. Once you have your final assembly, rename your assembly directory int `spades-final`, e.g. `mv assembly/spades-default assembly/spades-final`.
2. Write down in your notes the command used to create your final assembly.
3. Write down in your notes the assembly statistics derived through Quast[162]

4.9 Further reading

4.9.1 Background on Genome Assemblies

- How to apply de Bruijn graphs to genome assembly. *[COMPEAU2011]* (page 85)
- Sequence assembly demystified. *[NAGARAJAN2013]* (page 85)

4.9.2 Evaluation of Genome Assembly Software

- GAGE: A critical evaluation of genome assemblies and assembly algorithms. *[SALZBERG2012]* (page 85)
- Assessment of de novo assemblers for draft genomes: a case study with fungal genomes. *[ABBAS2014]* (page 85)

4.10 Web links

- Lectures for this topic: Genome Assembly: An Introduction[163]
- SPAdes[164]

[160] http://quast.bioinf.spbau.ru/
[161] http://bioinf.spbau.ru/spades
[162] http://quast.bioinf.spbau.ru/
[163] https://dx.doi.org/10.6084/m9.figshare.2972323.v1
[164] http://bioinf.spbau.ru/spades

- Quast[165]
- Bandage[166] (Bioinformatics Application for Navigating De novo Assembly Graphs Easily) is a program that visualizes a genome assembly as a graph *[WICK2015]* (page 86).

[165] http://quast.bioinf.spbau.ru/
[166] 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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
```

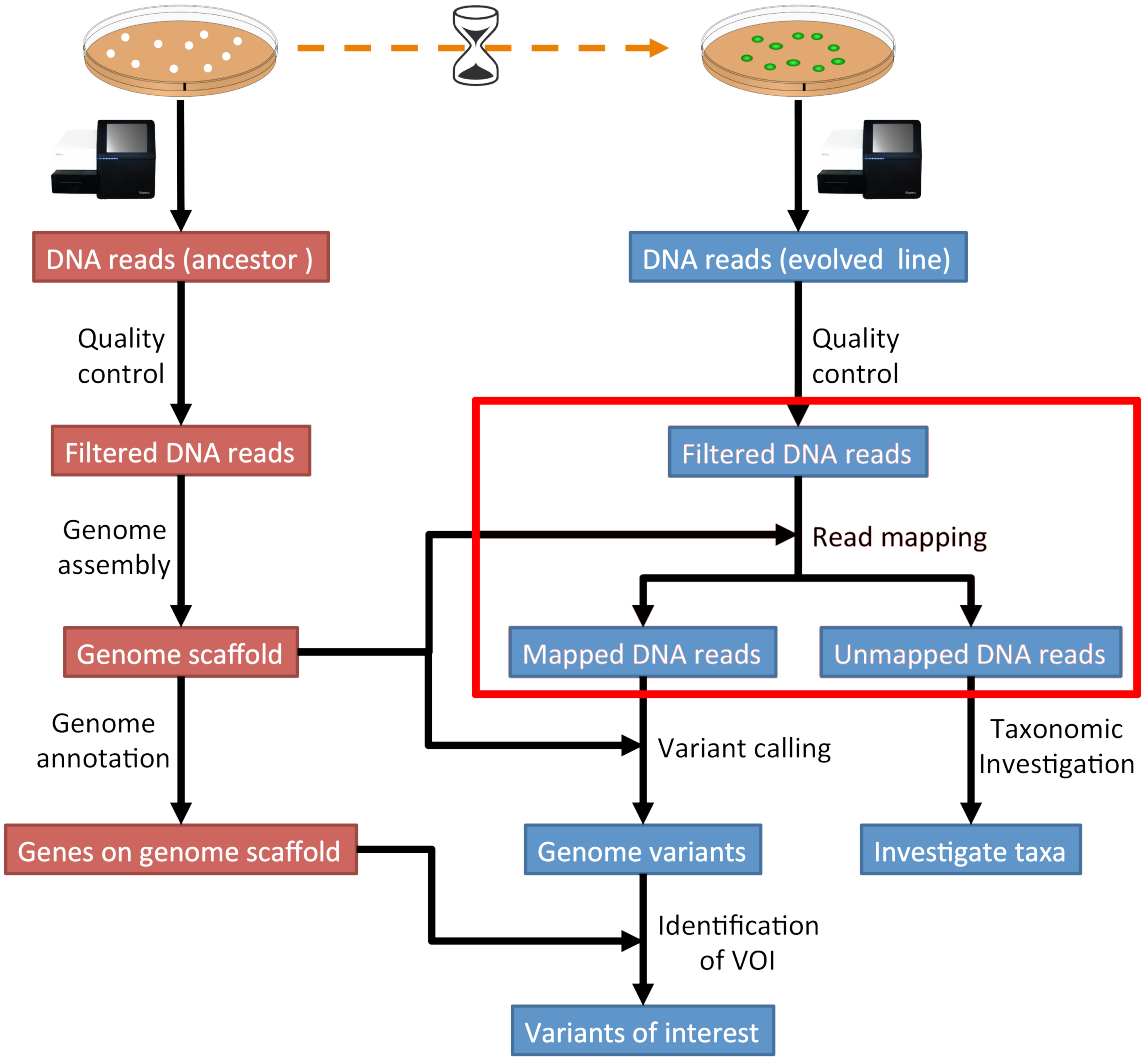


Fig. 5.1: The part of the workflow we will work on in this section marked in red.

5.5 Mapping sequence reads to a reference genome

We want to map the sequencing reads to the ancestral reference genome we created in the section *Genome assembly* (page 23). We are going to use the quality trimmed forward and backward DNA sequences of the evolved line and use a program called BWA[200] 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 Installing the software

We are going to use a program called BWA[201] fo map our reads to a genome.

It is simple to install and use.

```
conda activate ngs
conda install samtools
conda install bamtools
conda install bedtools
conda install bowtie2
conda install bwa
conda install picard
```

5.6 Bowtie2

5.6.1 Overview

Bowtie2[202] is a short read aligner, that can take a reference genome and map single- or paired-end data to it *[TRAPNELL2009]* (page 86). It requires an indexing step in which one supplies the reference genome and Bowtie2[203] will create an index that in the subsequent steps will be used for aligning the reads to the reference genome. The general command structure of the Bowtie2[204] tools we are going to use are shown below:

```
# bowtie2 help
bowtie2-build

# indexing
bowtie2-build genome.fasta PATH_TO_INDEX_PREFIX

# paired-end mapping
bowtie2 -X 1000 -x PATH_TO_INDEX_PREFIX -1 read1.fq.gz -2 read2.fq.gz -S aln-pe.sam
```

- `-X`: Adjust the maximum fragment size (length of paired-end alignments + insert size) to 1000bp. This might be useful if you do not know the exact insert size of your data. The Bowtie2[205] default is set to 500 which is often considered too short[206].

[200] http://bio-bwa.sourceforge.net/
[201] http://bio-bwa.sourceforge.net/
[202] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[203] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[204] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[205] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[206] http://lab.loman.net/2013/05/02/use-x-with-bowtie2-to-set-minimum-and-maximum-insert-sizes-for-nextera-libraries/

5.6.2 Creating a reference index for mapping

Todo: Create an Bowtie2[207] index for our reference genome assembly. Attention! Remember which file you need to submit to Bowtie2[208].

Hint: Should you not get it right, try the commands in *Code: Bowtie2 indexing* (page 78).

Note: Should you be unable to run Bowtie2[209] indexing on the data, you can download the index from *Downloads* (page 79). Unarchive and uncompress the files with `tar -xvzf bowtie2-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 filtered and trimmed sequencing reads of our evolved line to the reference genome.

Todo: Use the correct `bowtie2` command structure from above and map the reads of the evolved line to the reference genome.

Hint: Should you not get it right, try the commands in *Code: Bowtie2 mapping* (page 78).

Note: Bowtie2[210] does give very cryptic error messages without telling much why it did not want to run. The most likely reason is that you specified the paths to the files and result file wrongly. Check this first. Use tab completion a lot!

5.7 BWA

Attention: If the mapping did not succeed with Bowtie2[211]. We can use the aligner BWA[212] explained in this section. If the mapping with Bowtie2[213] did work, you can jump this section.

5.7.1 Overview

BWA[214] is a short read aligner, that can take a reference genome and map single- or paired-end data to it. It requires an indexing step in which one supplies the reference genome and BWA[215] will create an index that in the subsequent steps will be used for aligning the reads to the reference genome. The general command structure of the BWA[216] tools we are going to use are shown below:

[207] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[208] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[209] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[210] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[211] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[212] http://bio-bwa.sourceforge.net/
[213] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[214] http://bio-bwa.sourceforge.net/
[215] http://bio-bwa.sourceforge.net/
[216] http://bio-bwa.sourceforge.net/

```
# bwa index help
bwa index

# indexing
bwa index reference-genome.fa

# bwa mem help
bwa mem

# single-end mapping
bwa mem reference-genome.fa reads.fq > aln-se.sam

# paired-end mapping
bwa mem reference-genome.fa read1.fq read2.fq > aln-pe.sam
```

5.7.2 Creating a reference index for mapping

Todo: Create an BWA[217] index for our reference genome assembly. Attention! Remember which file you need to submit to BWA[218].

Hint: Should you not get it right, try the commands in *Code: BWA indexing* (page 78).

Note: Should you be unable to run BWA[219] indexing on the data, you can download the index from *Downloads* (page 79). Unarchive and uncompress the files with `tar -xvzf bwa-index.tar.gz`.

5.7.3 Mapping reads in a paired-end manner

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

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

Hint: Should you not get it right, try the commands in *Code: BWA mapping* (page 78).

5.8 The sam mapping file-format

Bowtie2[220] and BWA[221] 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[222] and even more

[217] http://bio-bwa.sourceforge.net/
[218] http://bio-bwa.sourceforge.net/
[219] http://bio-bwa.sourceforge.net/
[220] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[221] http://bio-bwa.sourceforge.net/
[222] http://bio-bwa.sourceforge.net/bwa.shtml#4

information can be found here[223]. 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
↪FHGHHHHHGGGHHHHHHHHHHHHHHHHHHGHHHHHHHHHHHHGHHHHGHHHHHHHHGDHHHHHHHHGHHHHGHHHGHHHHHHFHHHHGHHHHIHHHHHHHHHHHHHHHHHGHH
↪NM:i:0  MD:Z:151        AS:i:151        XS:i:0
```

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

5.9 Mapping post-processing

5.9.1 Fix mates and compress

Because aligners can sometimes leave unusual SAM flag[224] information on SAM records, it is helpful when working with many tools to first clean up read pairing information and flags with SAMtools[225]. We are going to produce also compressed bam output for efficient storing of and access to the mapped reads.

```
samtools fixmate -O bam mappings/evolved-6.sam mappings/evolved-6.fixmate.bam
```

- `-O bam`: specifies that we want compressed bam output

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[226] information later below to extract specific alignments.

[223] http://samtools.github.io/hts-specs/SAMv1.pdf
[224] http://bio-bwa.sourceforge.net/bwa.shtml#4
[225] http://samtools.sourceforge.net/
[226] http://bio-bwa.sourceforge.net/bwa.shtml#4

Hint: A very useful tools to explain flags can be found here[227].

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

```
rm mappings/evolved-6.sam
```

5.9.2 Sorting

We are going to use SAMtools[228] again to sort the bam-file into coordinate order:

```
# convert to bam file and sort
samtools sort -O bam -o mappings/evolved-6.sorted.bam mappings/evolved-6.fixmate.bam
```

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

5.9.3 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.

```
picard MarkDuplicates REMOVE_DUPLICATES=true METRICS_FILE=mappings/evolved-6.marked_dup_metrics.txt␣
↪INPUT=mappings/evolved-6.sorted.bam OUTPUT=mappings/evolved-6.sorted.dedup.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 79).

5.10 Mapping statistics

5.10.1 Stats with SAMtools

Lets get an mapping overview:

```
samtools flagstat mappings/evolved-6.sorted.dedup.bam
```

Todo: Look at the mapping statistics and understand their meaning[229]. 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?

[227] http://broadinstitute.github.io/picard/explain-flags.html
[228] http://samtools.sourceforge.net/
[229] https://www.biostars.org/p/12475/

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/evolved-6.sorted.dedup.bam | gzip > mappings/evolved-6.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/evolved-6.depth.txt.gz | egrep '^NODE_20_' | gzip >  mappings/NODE_20.depth.txt.gz
```

Now we quickly use some R[230] to make a coverage plot for contig NODE20. Open a R[231] 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

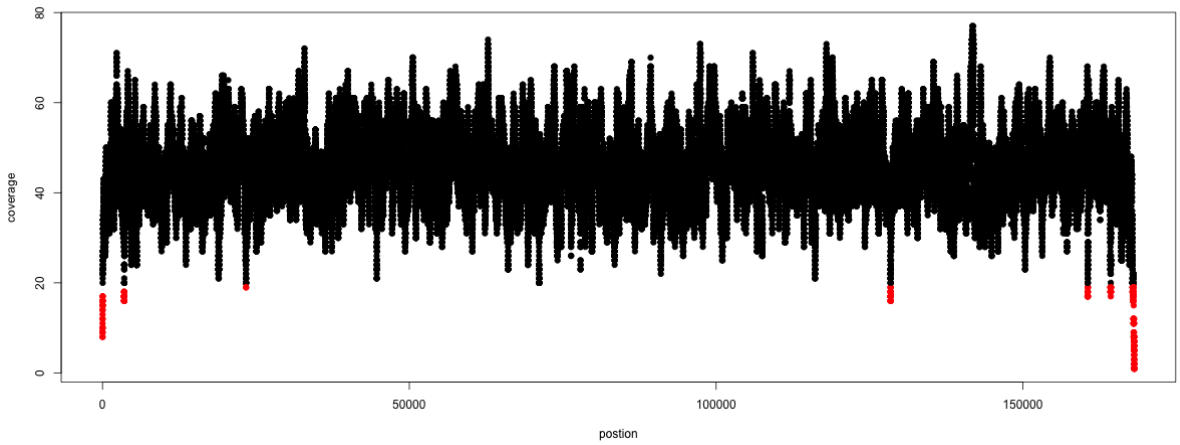


Fig. 5.2: A example coverage plot for a contig with highlighted in red regions with a coverage below 20 reads.

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.

[230] https://www.r-project.org/
[231] https://www.r-project.org/

5.10.2 Stats with QualiMap

For a more in depth analysis of the mappings, one can use QualiMap[232].

QualiMap[233] 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.

Installation:

```
conda install qualimap
```

Run QualiMap[234] with:

```
qualimap bamqc -bam mappings/evolved-6.sorted.dedup.bam
```

This will create a report in the mapping folder. See this webpage[235] to get help on the sections in the report.

Todo: Install QualiMap[236] and investigate the mapping of the evolved sample. Write down your observations.

5.11 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[237] 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[238].

Todo: Explain what concordant and discordant read pairs are? Look at the Bowtie2[239] manual.

5.11.1 Concordant reads

We select read-pair that have been mapped in a correct manner (same chromosome/contig, correct orientation to each other).

```
samtools view -h -b -f 3 mappings/evolved-6.sorted.dedup.bam > mappings/evolved-6.sorted.dedup.
↪concordant.bam
```

- `-h`: Include the sam header
- `-b`: Output will be bam-format
- `-f 3`: Only extract correctly paired reads. `-f` extracts alignments with the specified SAM flag[240] set.

[232] http://qualimap.bioinfo.cipf.es/
[233] http://qualimap.bioinfo.cipf.es/
[234] http://qualimap.bioinfo.cipf.es/
[235] http://qualimap.bioinfo.cipf.es/doc_html/analysis.html#output
[236] http://qualimap.bioinfo.cipf.es/
[237] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[238] http://samtools.sourceforge.net/
[239] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[240] http://bio-bwa.sourceforge.net/bwa.shtml#4

5.11.2 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[241]. 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[242]:

```
samtools view -h -b -q 20 mappings/evolved-6.sorted.dedup.concordant.bam > mappings/evolved-6.
↪sorted.dedup.concordant.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[243] 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[244] 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.

5.11.3 Unmapped reads

We could decide to use Kraken[245] like in section *Taxonomic investigation* (page 39) 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/evolved-6.sorted.dedup.bam > mappings/evolved-6.sorted.unmapped.bam

# count them
samtools view -c mappings/evolved-6.sorted.unmapped.bam
```

- `-b`: indicates that the output is BAM.
- `-f INT`: only include reads with this SAM flag[246] 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.

```
bamToFastq -i mappings/evolved-6.sorted.unmapped.bam -fq mappings/evolved-6.sorted.unmapped.R1.
↪fastq -fq2  mappings/evolved-6.sorted.unmapped.R2.fastq
```

[241] https://sequencing.qcfail.com/articles/mapq-values-are-really-useful-but-their-implementation-is-a-mess/
[242] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
[243] https://sequencing.qcfail.com/articles/mapq-values-are-really-useful-but-their-implementation-is-a-mess/
[244] http://qualimap.bioinfo.cipf.es/
[245] https://ccb.jhu.edu/software/kraken/
[246] http://bio-bwa.sourceforge.net/bwa.shtml#4

CHAPTER

SIX

TAXONOMIC INVESTIGATION

6.1 Preface

We want to investate 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 Kraken[275] 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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
```

[275] https://ccb.jhu.edu/software/kraken/

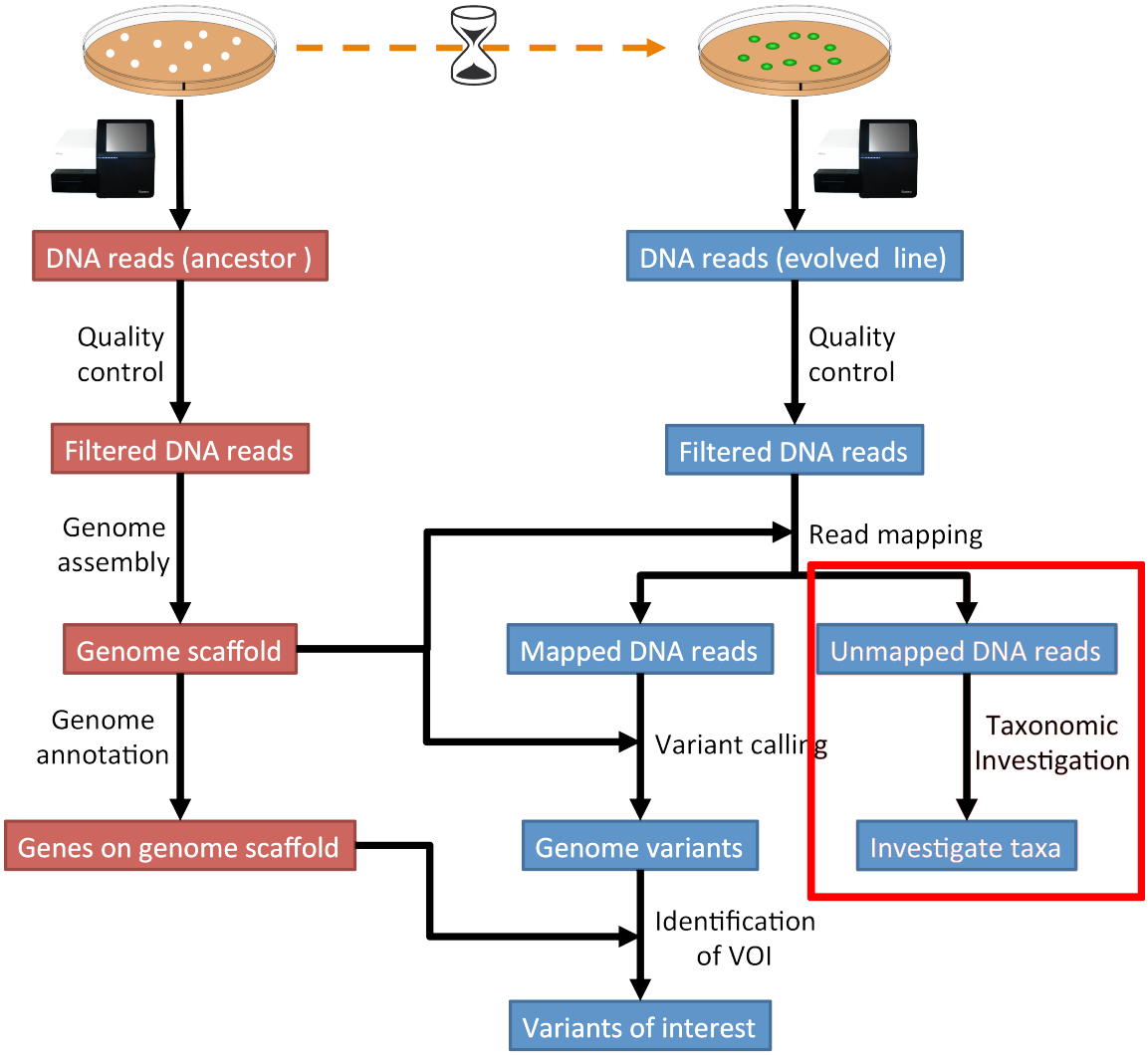


Fig. 6.1: The part of the workflow we will work on in this section marked in red.

6.4 Kraken

We will be using a tool called Kraken[276] *[WOOD2014]* (page 86). This tool uses k-mers to assign a taxonomic labels in form of NCBI Taxonomy[277] 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 Kraken[278]:

```
conda activate ngs
conda install kraken-all
```

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 Kraken[279] database that can be used to assign the taxonomic labels to sequences. We opt for downloading the pre-build "minikraken" database from the Kraken[280] website:

```
curl -O https://ccb.jhu.edu/software/kraken/dl/minikraken.tgz

# alternatively we can use wget
wget https://ccb.jhu.edu/software/kraken/dl/minikraken.tgz

# once the download is finished, we need to extract the archive content
# it will create a directory: "minikraken_20141208/"
tar -xvzf minikraken.tgz
```

Attention: Should the download fail. Please find links to alternative locations on the *Downloads* (page 79) page.

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

6.4.2 Usage

Now that we have installed Kraken[281] and downloaded and extracted the minikraken database, we can attempt to investigate the sequences we got back from the sequencing provider for other species as

[276] https://ccb.jhu.edu/software/kraken/
[277] https://www.ncbi.nlm.nih.gov/taxonomy
[278] https://ccb.jhu.edu/software/kraken/
[279] https://ccb.jhu.edu/software/kraken/
[280] https://ccb.jhu.edu/software/kraken/
[281] https://ccb.jhu.edu/software/kraken/

the one it should contain. We call the Kraken[282] tool and specify the database and fasta-file with the sequences it should use. The general command structure looks like this:

```
kraken --only-classified-output --db minikraken_20141208 example.fa > example.kraken
```

However, we may have fastq-files, so we need to use `--fastq-input` which tells Kraken[283] that it is dealing with fastq-formated files. Here, we are investigating one of the unmapped paired-end read files of the evolved line.

```
kraken --only-classified-output --db minikraken_20141208 --fastq-input ../mappings/evolved-6.sorted.
↪unmapped.R1.fastq > evolved-6-R1.kraken
```

Note: We are for now only interested in the portion of sequeuces that can be classified, that is why we supplied the option `--only-classified-output` to the Kraken[284] command.

This classification may take a while, depending on how many sequences we are going to classify. The resulting content of the file "evolved-6-R1.kraken" looks similar to the following example:

```
C       M02810:197:000000000-AV55U:1:1101:10078:18384/1 2157    151     0:99 2157:1 0:1␣
↪2157:1 0:19
C       M02810:197:000000000-AV55U:1:1101:10573:27304/1 364745  150     0:43 364745:1 0:76
C       M02810:197:000000000-AV55U:1:1101:10852:5722/1  37665   151     0:33 37665:1 0:87
C       M02810:197:000000000-AV55U:1:1101:11429:10330/1 374840  101     0:17 374840:1 0:4␣
↪374840:1 0:2 374840:1 0:45
C       M02810:197:000000000-AV55U:1:1101:11705:24355/1 2157    151     0:1 2157:1 0:119
```

Each sequence classified by Kraken[285] 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 Kraken used to label the sequence; this is **0** if the sequence is unclassified and otherwise should be the NCBI Taxonomy[286] 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

Note: The Kraken[287] manual can be accessed here[288].

---

[282] https://ccb.jhu.edu/software/kraken/
[283] https://ccb.jhu.edu/software/kraken/
[284] https://ccb.jhu.edu/software/kraken/
[285] https://ccb.jhu.edu/software/kraken/
[286] https://www.ncbi.nlm.nih.gov/taxonomy
[287] https://ccb.jhu.edu/software/kraken/
[288] http://ccb.jhu.edu/software/kraken/MANUAL.html

6.4.3 Investigate taxa

We can use the webpage NCBI TaxIdentifier[289] to quickly get the names to the taxonomy identifier. However, this is impractical as we are dealing potentially with many sequences. Kraken[290] has some scripts that help us understand our results better.

kraken-report

First, we generate a sample-wide report of all taxa found. This can be achieved with the tool `kranken-report`.

```
kraken-report --db minikraken_20141208 evolved-6-R1.kraken > evolved-6-R1.kraken.report
```

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

```
  0.00  0       0       U       0       unclassified
100.00  2665    47      -       1       root
 54.56  1454    0       -       131567    cellular organisms
 38.50  1026    1023    D       2157        Archaea
  0.08  2       0       P       651137        Thaumarchaeota
  0.08  2       0       -       651142          unclassified Thaumarchaeota
  0.08  2       0       G       1048752           Candidatus Caldiarchaeum
  0.08  2       2       S       311458              Candidatus Caldiarchaeum subterraneum
  0.04  1       0       P       28890         Euryarchaeota
  0.04  1       0       C       183967          Thermoplasmata
```

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[291] 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 `--show-zeros` 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`.

```
kraken-report *--show-zeros* --db minikraken_20141208 evolved-6-R1.kraken | **sort -n -k5** >␣
↪evolved-6-R1.kraken.report.sorted
```

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[292] id.

[289] https://www.ncbi.nlm.nih.gov/Taxonomy/TaxIdentifier/tax_identifier.cgi
[290] https://ccb.jhu.edu/software/kraken/
[291] https://www.ncbi.nlm.nih.gov/taxonomy
[292] https://www.ncbi.nlm.nih.gov/taxonomy

kraken-translate

For every sequence in our sample and its predicted taxonomic identifier, we can attach the taxonomic names with `kraken-translate`.

```
kraken-translate --mpa-format --db minikraken_20141208 evolved-6-R1.kraken > evolved-6-R1.kraken.
↪names
```

An example output looks like this:

```
M02810:197:000000000-AV55U:1:1101:10078:18384/1 d__Archaea
M02810:197:000000000-AV55U:1:1101:10573:27304/1 d__Viruses|f__Baculoviridae|g__
↪Betabaculovirus|s__Choristoneura_occidentalis_granulovirus
M02810:197:000000000-AV55U:1:1101:10852:5722/1  d__Viruses|f__Phycodnaviridae|g__
↪Phaeovirus|s__Ectocarpus_siliculosus_virus_1
M02810:197:000000000-AV55U:1:1101:11429:10330/1 d__Viruses|f__Microviridae|g__
↪Microvirus|s__Enterobacteria_phage_phiX174_sensu_lato
M02810:197:000000000-AV55U:1:1101:11705:24355/1 d__Archaea
M02810:197:000000000-AV55U:1:1101:12206:13333/1 d__Viruses|o__Herpesvirales|f__
↪Alloherpesviridae|g__Cyprinivirus|s__Cyprinid_herpesvirus_3
M02810:197:000000000-AV55U:1:1101:12336:11626/1 d__Viruses|f__Baculoviridae|g__
↪Betabaculovirus|s__Choristoneura_occidentalis_granulovirus
M02810:197:000000000-AV55U:1:1101:12707:5809/1  d__Archaea
M02810:197:000000000-AV55U:1:1101:12741:21685/1 d__Viruses|f__Baculoviridae|g__
↪Betabaculovirus|s__Choristoneura_occidentalis_granulovirus
M02810:197:000000000-AV55U:1:1101:12767:27821/1 d__Archaea
```

Here, each sequence that got classified is present in one row. The nomenclature for the names is preceded with an letter according to its rank, e.g. **(d)omain, (k)ingdom, (p)hylum, (c)lass, (o)rder, (f)amily (g)enus, or (s)pecies**. Taxonomy assignments above the superkingdom (**d**) rank are represented as just **root**.

6.5 Centrifuge

We can also use the successor to Kraken[293], a tool by the same group called Centrifuge[294] *[KIM2017]* (page 86). 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[295] 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.

6.5.1 Installation

Use conda in the same fashion as before to install Centrifuge[296]:

```
conda activate ngs
conda install centrifuge
```

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

[293] https://ccb.jhu.edu/software/kraken/
[294] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[295] https://www.ncbi.nlm.nih.gov/taxonomy
[296] http://www.ccb.jhu.edu/software/centrifuge/index.shtml

```
# 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[297] database that can be used to assign the taxonomic labels to sequences. We opt for downloading the pre-build database from the Centrifuge[298] website:

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

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

Attention: Should the download fail. Please find links to alternative locations on the *Downloads* (page 79) 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[299] 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[300] tool and specify the database and fasta-file with the sequences it should use. The general command structure looks like this:

```
centrifuge -x p_compressed -U example.fa --report-file report.txt -S results.txt
```

However, if we do not have fastq-files we may have to use the `-f` option, which tells Centrifuge[301] that it is dealing with a fasta-formated file. Here, we are investigating one of the unmapped paired-end read files of the evolved line.

```
centrifuge -x p_compressed -U ../mappings/evolved-6.sorted.unmapped.R1.fastq --report-file evolved-
↪6-R1-report.txt -S evolved-6-R1-results.txt
```

This classification may take a moment, depending on how many sequences we are going to classify. The resulting content of the file `evolved-6-R1-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
```

[297] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[298] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[299] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[300] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[301] http://www.ccb.jhu.edu/software/centrifuge/index.shtml

```
M02810:197:000000000-AV55U:1:1101:19743:5166    cid|562 562     10404   10404   117     ␣
↪151    2
```

Each sequence classified by Centrifuge[302] results in a single line of output. Output lines contain eight tab-delimited fields; from left to right, they are according to the Centrifuge[303] 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.

6.5.3 Investigate taxa

Centrifuge report

The command above creates a Centrifuge[304] 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[305] 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 Kraken[306], we can do it like this:

[302] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[303] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[304] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[305] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[306] https://ccb.jhu.edu/software/kraken/

```
centrifuge-kreport -x p_compressed 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
```

This gives a similar (not the same) report as the Kraken[307] 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[308] tools to create a nice interactive visualisation of the taxa content of our sample *[ONDOV2011]* (page 86). Fig. 6.2 shows an example (albeit an artificial one) snapshot of the visualisation Krona[309] provides. Fig. 6.2 is a snapshot of the interactive web-page similar to the one we try to create.

6.6.1 Installation

Install Krona[310] with:

```
source activate ngs
conda install krona
```

[307] https://ccb.jhu.edu/software/kraken/
[308] https://github.com/marbl/Krona/wiki
[309] https://github.com/marbl/Krona/wiki
[310] https://github.com/marbl/Krona/wiki

Fig. 6.2: Example of an Krona output webpage.

First some house-keeping to make the Krona[311] 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
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[312]. 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
```

Now, if this fails, we download a pre-build taxonomy database for krona:

```
# Download pre-build database
curl -O http://compbio.massey.ac.nz/data/taxonomy.tab.gz

# 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
```

Attention: Should this also fail we can download a pre-build database on the *Downloads* (page 79) page via a browser.

[311] https://github.com/marbl/Krona/wiki
[312] https://github.com/marbl/Krona/wiki

6.6.3 Visualise

Now, we use the tool `ktImportTaxonomy` from the Krona[313] 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 Kraken[314] or Centrifuge[315] results:

```
# Kraken
cd kraken
cat evolved-6-R1.kraken | cut -f 2,3 > evolved-6-R1.kraken.krona
ktImportTaxonomy evolved-6-R1.kraken.krona
firefox taxonomy.krona.html

# Centrifuge
cd centrifuge
cat evolved-6-R1-results.txt | cut -f 1,3 > evolved-6-R1-results.krona
ktImportTaxonomy evolved-6-R1-results.krona
firefox taxonomy.krona.html
```

What happens here is that we extract the second and third column from the Kraken[316] results. Afterwards, we input these to the Krona[317] script, and open the resulting web-page in a bowser. Done!

[313] https://github.com/marbl/Krona/wiki
[314] https://ccb.jhu.edu/software/kraken/
[315] http://www.ccb.jhu.edu/software/centrifuge/index.shtml
[316] https://ccb.jhu.edu/software/kraken/
[317] 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]* (page 86).

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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
```

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.

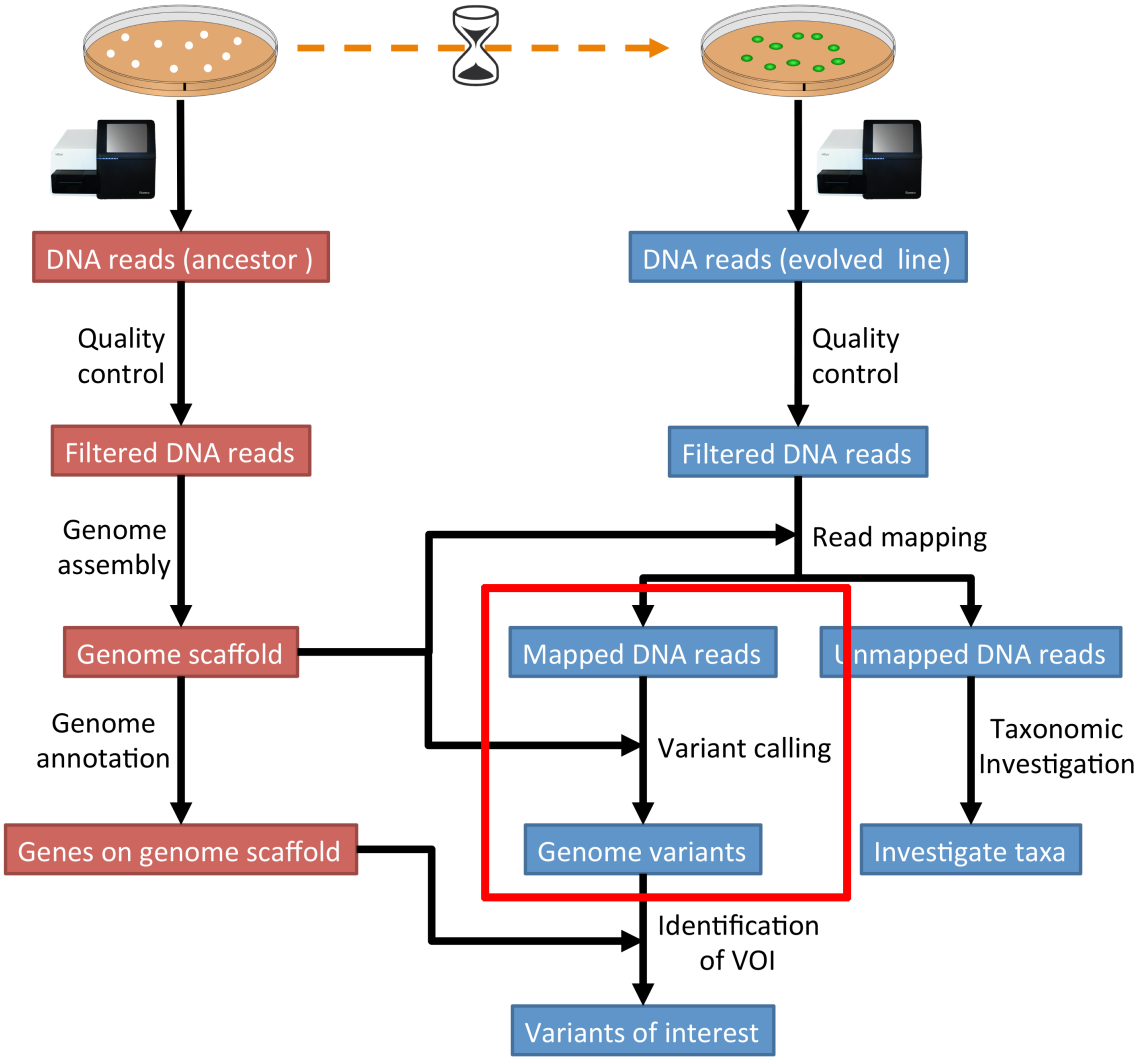


Fig. 7.1: The part of the workflow we will work on in this section marked in red.

```
# activate the env
conda activate ngs

# Install these tools into the conda environment
# if not already installed
conda install samtools
conda install bamtools
conda install freebayes
conda install bedtools
conda install vcflib
conda install rtg-tools
conda install bcftools
```

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/spades_final`, use SAMtools[348] to do this:

```
samtools faidx assembly/spades-final/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/evolved-6.sorted.concordant.q20.bam
```

Lets also create a new directory for the variants:

```
mkdir variants
```

7.7 Calling variants

7.7.1 SAMtools mpileup

We use the sorted filtered bam-file that we produced in the mapping step before.

```
# We first pile up all the reads and then call variants
samtools mpileup -u -g -f assembly/spades-final/scaffolds.fasta mappings/evolved-6.sorted.
↪concordant.q20.bam | bcftools call -v -m -O z -o variants/evolved-6.mpileup.vcf.gz
```

SAMtools[349] mpileup parameter:

- `-u`: uncompressed output
- `-g`: generate genotype likelihoods in BCF format
- `-f FILE`: faidx indexed reference sequence file

BCFtools[350] view parameter:

- `-v`: output variant sites only
- `-m`: alternative model for multiallelic and rare-variant calling
- `-o`: output file-name

[348] http://samtools.sourceforge.net/
[349] http://samtools.sourceforge.net/
[350] http://www.htslib.org/doc/bcftools.html

- `-O z`: output type: 'z' compressed VCF

7.7.2 Freebayes

As an alternative we can do some variant calling with another tool called freebayes[351]. 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[352] like so:

```
# Now we call variants and pipe the results into a new file
freebayes -f assembly/spades-final/scaffolds.fasta mappings/evolved-6.sorted.concordant.q20.bam |␣
↪gzip > variants/evolved-6.freebayes.vcf.gz
```

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
zcat variants/evolved-6.mpileup.vcf.gz | head
```

```
##fileformat=VCFv4.2
##FILTER=<ID=PASS,Description="All filters passed">
##samtoolsVersion=1.3.1+htslib-1.3.1
##samtoolsCommand=samtools mpileup -g -f assembly/spades-final/scaffolds.fasta -o variants/evolved-
↪6.mpileup.bcf mappings/evolved-6.sorted.concordant.q20.bam
##reference=file://assembly/spades-final/scaffolds.fasta
##contig=<ID=NODE_1_length_1419525_cov_15.3898,length=1419525>
##contig=<ID=NODE_2_length_1254443_cov_15.4779,length=1254443>
##contig=<ID=NODE_3_length_972329_cov_15.3966,length=972329>
##contig=<ID=NODE_4_length_951685_cov_15.4231,length=951685>
##contig=<ID=NODE_5_length_925222_cov_15.39,length=925222>
##contig=<ID=NODE_6_length_916533_cov_15.4426,length=916533>
```

Lets look at the variants:

```
# remove header lines and look at top 4 entires
zcat variants/evolved-6.mpileup.vcf.gz | egrep -v '##' | head -4
```

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  mappings/evolved-6.sorted.
↪concordant.q20.bam
NODE_1_length_1419525_cov_15.3898       24721   .       T       C       164     .       DP=12;VDB=0.
↪205941;SGB=-0.680642;MQ0F=0;AC=2;AN=2;DP4=0,0,12,0;MQ=40      GT:PL   1/1:191,36,0
NODE_1_length_1419525_cov_15.3898       157033  .       AAGAGAGAGAGAGAGAGAGAGAGA        ␣
↪AAGAGAGAGAGAGAGAGAGA   39.3328  .       INDEL;IDV=6;IMF=0.146341;DP=41;VDB=0.0813946;SGB=-0.
↪616816;MQSB=1;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=13,17,3,3;MQ=42      GT:PL   0/1:75,0,255
NODE_1_length_1419525_cov_15.3898       162469  .       T       C       19.609  .       DP=16;VDB=0.
↪045681;SGB=-0.511536;RPB=0.032027;MQB=0.832553;BQB=0.130524;MQ0F=0;ICB=1;HOB=0.5;AC=1;AN=2;DP4=13,
↪0,3,0;MQ=39        GT:PL   0/1:54,0,155
```

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

[351] https://github.com/ekg/freebayes
[352] https://github.com/ekg/freebayes

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`:

```
tabix -p vcf variants/evolved-6.mpileup.vcf.gz
```

- `-p vcf`: input format

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

```
rtg vcfstats variants/evolved-6.mpileup.vcf.gz
```

Example output from `rtg vcfstats`:

```
Location                   : variants/evolved-6.mpileup.vcf.gz
Failed Filters             : 0
Passed Filters             : 516
SNPs                       : 399
MNPs                       : 0
Insertions                 : 104
Deletions                  : 13
Indels                     : 0
Same as reference          : 0
SNP Transitions/Transversions: 1.87 (286/153)
Total Het/Hom ratio        : 3.20 (393/123)
SNP Het/Hom ratio          : 8.98 (359/40)
MNP Het/Hom ratio          : - (0/0)
Insertion Het/Hom ratio    : 0.30 (24/80)
Deletion Het/Hom ratio     : 3.33 (10/3)
Indel Het/Hom ratio        : - (0/0)
Insertion/Deletion ratio   : 8.00 (104/13)
Indel/SNP+MNP ratio        : 0.29 (117/399)
```

However, we can also run BCFtools[353] to extract more detailed statistics about our variant calls:

```
bcftools stats -F assembly/spades-final/scaffolds.fasta -s - variants/evolved-6.mpileup.vcf.gz >␣
↪variants/evolved-6.mpileup.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:

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

```
mkdir variants/plots
plot-vcfstats -p variants/plots/ variants/evolved-6.mpileup.vcf.gz.stats
```

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

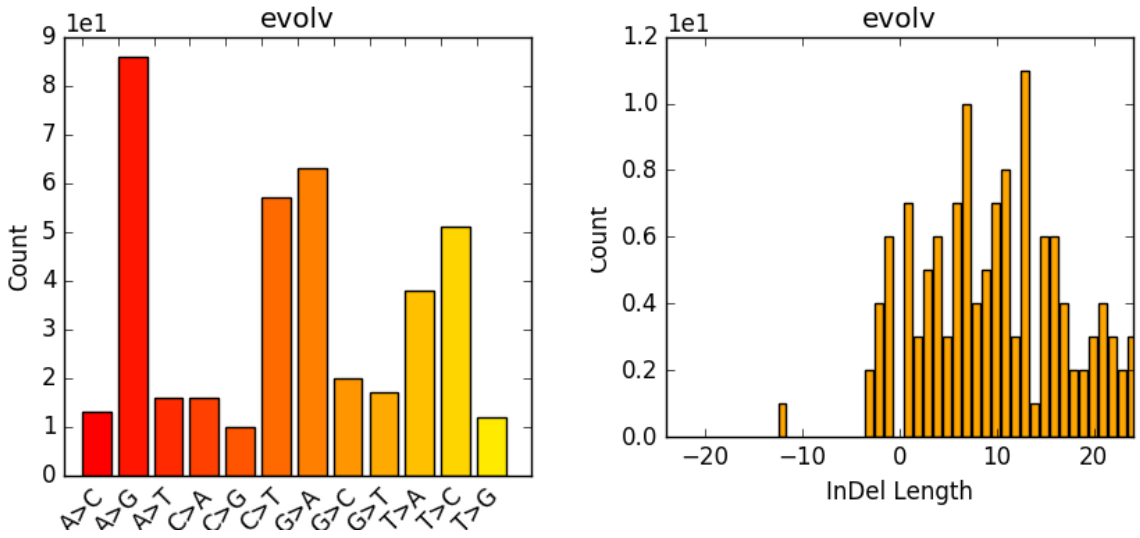


Fig. 7.2: Example of `plot-vcfstats` output.

7.8.3 Variant filtration

Variant filtration is a big topic in itself *[OLSEN2015]* (page 86). 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/evolved-6.mpileup.vcf.gz -o variants/evolved-6.mpileup.q30.vcf.gz
```

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

or use vcflib[354]:

```
# or use vcflib
zcat variants/evolved-6.mpileup.vcf.gz  | vcffilter -f "QUAL >= 30" | gzip > variants/evolved-6.
↪mpileup.q30.vcf.gz z
```

- `-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/evolved-6.mpileup.q30.vcf.gz
```

[354] https://github.com/vcflib/vcflib#vcflib

freebayes[355] adds some extra information to the vcf-fiels it creates. This allows for some more detailed filtering. This strategy will NOT work on the SAMtools[356] mpileup called variants Here we filter, based on some recommendation form the developer of freebayes[357]:

```
zcat variants/evolved-6.freebayes.vcf.gz  | vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR␣
↪> 0 & RPR > 1 & RPL > 1" | gzip > variants/evolved-6.freebayes.filtered.vcf.gz
```

- `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

---

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.

This strategy used here will do for our purposes. However, several more elaborate filtering strategies have been explored, e.g. here[358].

[355] https://github.com/ekg/freebayes
[356] http://samtools.sourceforge.net/
[357] https://github.com/ekg/freebayes
[358] 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[388] and BUSCO[389].

Attention: The annotation process will take up to 90 minutes. Start it as soon as possible.

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/
SolexaQA/
```

(continues on next page)

[388] http://augustus.gobics.de
[389] http://busco.ezlab.org

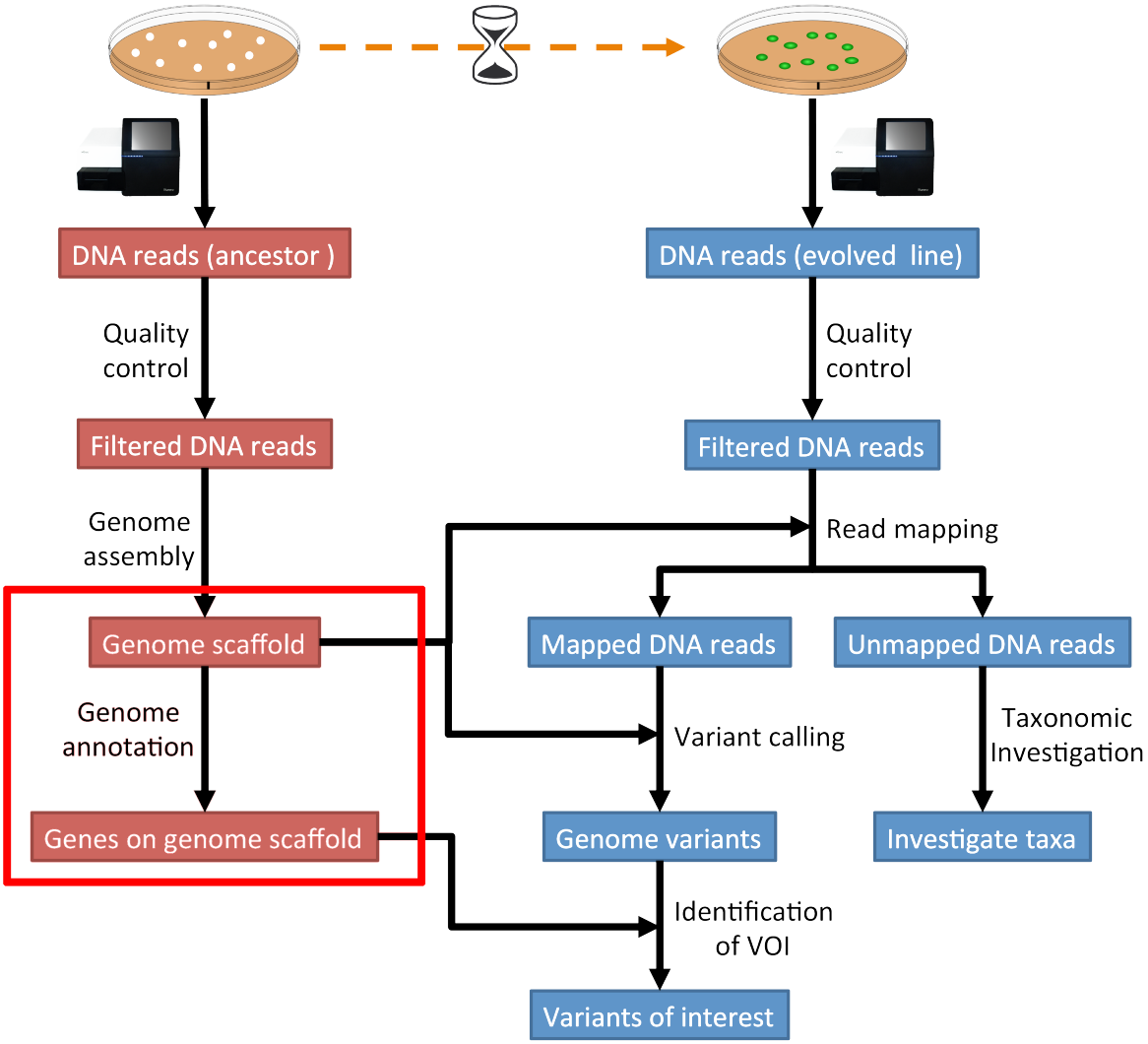


Fig. 8.1: The part of the workflow we will work on in this section marked in red.

(continued from previous page)

```
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
variants/
```

8.5 Installing the software

```
# activate the env
conda activate ngs

conda install busco
```

This will install both the Augustus[390] *[STANKE2005]* (page 86) and the BUSCO[391] *[SIMAO2015]* (page 86) 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[392] will use to assess orthologue presence absence in our genome annotation. We will use wget for this:

```
wget http://busco.ezlab.org/datasets/saccharomycetales_odb9.tar.gz

# unpack the archive
tar -xzvf saccharomycetales_odb9.tar.gz
```

Note: Should the download fail, download manually from *Downloads* (page 79).

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/ngs/config/ ./
```

We next need to specify the `path` to this config file so that the program knows where to look now that we have changed the location (note that this is all one line below):

```
export AUGUSTUS_CONFIG_PATH="~/analysis/annotation/config/"n
```

We next check that we have actually changed the `path` correctly. Entering this into the command should result in the file location being output on the next line of the command prompt.

```
echo $AUGUSTUS_CONFIG_PATH
```

[390] http://augustus.gobics.de
[391] http://busco.ezlab.org
[392] http://busco.ezlab.org

8.6 Assessment of orthologue presence and absence

BUSCO[393] will assess orthologue presence absence using blastn[394], a rapid method of finding close matches in large databases (we will discuss this in lecture). It uses blastn[395] 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).

```
busco -i ../assembly/spades-final/scaffolds.fasta -o file_name_of_your_choice -l ./
↪saccharomycetales_odb9 -m geno
```

Note: This should take about 90 minutes to run. So in the meantime do the next step.

8.7 Annotation

We will use Augustus[396] 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 S. cerevisiae)
- 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=saccharomyces_cerevisiae_S288C ../assembly/spades-
↪final/scaffolds.fasta > your_new_fungus.gff
```

Note: Should the process of producing your annotation fail, you can download a annotation manually from *Downloads* (page 79). Remember to unzip the file.

8.8 Interactive viewing

We will use the software IGV[397] to view the assembly, the gene predictions you have made, and the variants that you have called, all in one window.

[393] http://busco.ezlab.org
[394] https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch
[395] https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch
[396] http://augustus.gobics.de
[397] http://software.broadinstitute.org/software/igv/

8.9 Installing IGV[398]

We will not install this software using conda[399]. Instead, make a new directory in your home directory entitled "software", and change into this directory. You will have to download the software from the Broad Institute:

```
mkdir software
cd software
wget http://data.broadinstitute.org/igv/projects/downloads/2.4/IGV_2.4.10.zip

# unzip the software:
unzip IGV_2.4.10.zip

# and change into that directory.
cd IGV_2.4.10.zip

# To run the interactive GUI, you will need to run the bash script in that directory:
bash igv.sh
```

Note: Should the download fail, download manually from *Downloads* (page 79).

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

- Genome -> load Genome from File
- Load your assembly, not your gff file.

Load the tracks:

- File -> Load from file
- Load your `vcf` file from last week
- Load your `gff` file from this week.

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 yeast populations.

8.10 Assessment of orthologue presence and absence (2)

Hopefully your BUSCO[400] analysis will have finished by this time. 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.

[398] http://software.broadinstitute.org/software/igv/
[399] http://conda.pydata.org/miniconda.html
[400] http://busco.ezlab.org

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

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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
variants/
```

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

```
mkdir phylogeny
```

Download the fasta file of the *S. cerevisiase* TEF2 gene to the phylogeny folder:

```
cd phylogeny
curl -O http://compbio.massey.ac.nz/data/203341/s_cerev_tef2.fas
```

Note: Should the download fail, download manually from *Downloads* (page 79).

9.4 Installing the software

```
# activate the env
conda activate ngs

conda install blast
```

This will install a BLAST[430] executable that you can use to remotely query the NCBI database.

```
conda install muscle
```

This will install MUSCLE[431], alignment program that you can use to align nucleotide or protein sequences.

We will also install RAxML-NG[432], a phylogenetic tree inference tool, which uses maximum-likelihood (ML) optimality criterion. However, there is no conda repository for it yet. Thus, we need to download it manually.

```
wget
https://github.com/amkozlov/raxml-ng/releases/download/0.5.1/raxml-ng_v0.5.1b_linux_x86_64.zip

unzip raxml-ng_v0.5.1b_linux_x86_64.zip

rm raxml-ng_v0.5.1b_linux_x86_64.zip
```

9.5 Finding orthologues using BLAST

We will first make a BLAST[433] database of our current assembly so that we can find the orthologous sequence of the *S. cerevisiae* gene. To do this, we run the command `makeblastdb`:

```
# create blast db
makeblastdb in ../assembly/spades-final/scaffolds.fasta dbtype nucl
```

To run BLAST[434], we give it:

- `-db`: The name of the database that we are BLASTing
- `-query`: A fasta format input file
- A name for the output files
- Some notes about the format we want

First, we blast without any formatting:

```
blastn db ../assembly/spades-final/scaffolds.fasta query s_cerev_tef2.fas > blast.out
```

This should output a file with a set of BLAST[435] hits similar to what you might see on the BLAST[436] web site.

[430] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[431] http://www.ebi.ac.uk/Tools/msa/muscle/
[432] https://github.com/amkozlov/raxml-ng
[433] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[434] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[435] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[436] https://blast.ncbi.nlm.nih.gov/Blast.cgi

Read through the output (e.g. using nano) to see what the results of your BLAST[437] run was.

Next we will format the output a little so that it is easier to deal with.

```
blastn db ../assembly/spades-final/scaffolds.fasta query s_cerev_tef2.fas evalue 1e-100 outfmt "6␣
↪length sseq" > blast_formatted.out
```

This will yield a file that has only the sequences of the subject, so that we can later add those to other fasta files. However, the formatting is not perfect. To adjust the format such that it is fasta format, open the file in an editor (e.g. nano) and edit the first line so that it has a name for your sequence. You should know the general format of a fasta-file (e.g. the first line start with a ">").

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).

Next, you have to replace the dashes (signifying indels in the BLAST[438] result). This can easily be done in `vi`: Press the escape key, followed by: `:%s/\-//g`

Now we will BLAST[439] a remote database to get a list of hits that are already in the NCBI database.

Note: It turns out you may not be able to access this database from within BioLinux. In such a case, download the file named `blast.fas` and place it into your `~/analysis/phylogeny/` directory.

```
curl -O http://compbio.massey.ac.nz/data/203341/blast_u.fas
```

Append the fasta file of your yeast sequence to this file, using whatever set of commands you wish/know.

Note: Should the download fail, download manually from *Downloads* (page 79).

9.6 Performing an alignment

We will use MUSCLE[440] to perform our alignment on all the sequences in the BLAST[441] fasta file. This syntax is very simple (change the filenames accordingly):

```
muscle in infile.fas out your_alignment.aln
```

9.7 Building a phylogeny

We will use RAxML-NG[442] to build our phylogeny. This uses a maximum likelihood method to infer parameters of evolution and the topology of the tree. Again, the syntx of the command is fairly simple, except you must make sure that you are using the directory in which RAxML-NG[443] sits.

The arguments are:

- `-s`: an alignment file

[437] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[438] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[439] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[440] http://www.ebi.ac.uk/Tools/msa/muscle/
[441] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[442] https://github.com/amkozlov/raxml-ng
[443] https://github.com/amkozlov/raxml-ng

- `-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 your_alignment.aln -m GTRGAMMA n yeast_tree p 12345
```

9.8 Visualizing the phylogeny

We will use the online software Interactive Tree of Life (iTOL)[444] 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 yeast taxa is. To find out the closest relative, you will have to use the NCBI taxa page[445].

Todo: Are you certain that the yeast are related in the way that the phylogeny suggests? Why might the topology of this phylogeny not truly reflect the evolutionary history of these yeast species?

---

[444] http://itol.embl.de/upload.cgi
[445] 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/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
variants/
```

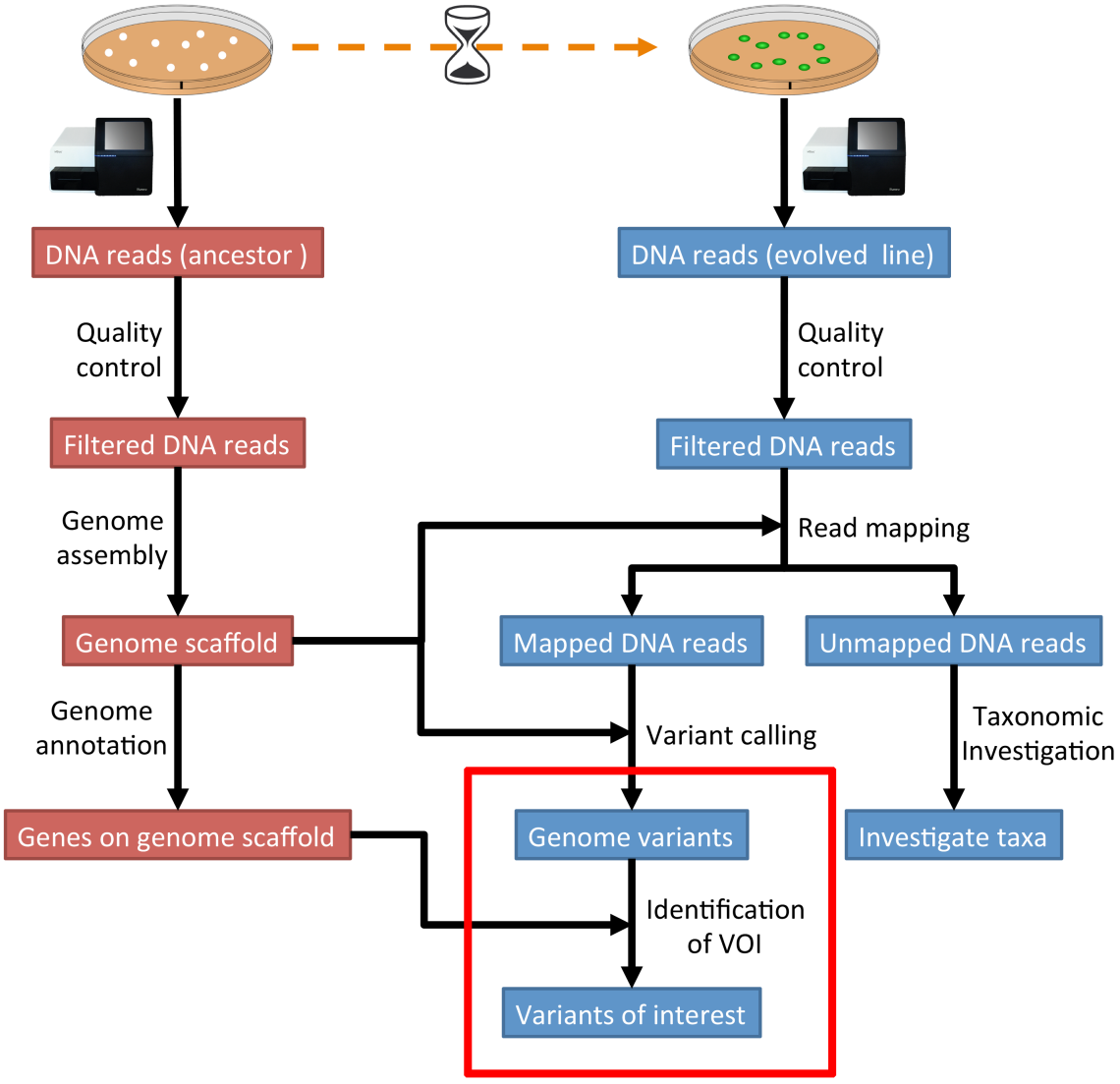


Fig. 10.1: The part of the workflow we will work on in this section marked in red.

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[473] 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.

```
# activate the env
conda activate ngs

# Install these tools into the conda environment
# if not already installed
conda install snpeff
conda install genometools-genometools
```

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

```
mkdir voi

# change into the directory
cd voi
```

10.6.2 Prepare SnpEff database

We need to create our own config-file for SnpEff[474]. Where is the `snpEff.config`:

```
find ~ -name snpEff.config
/home/manager/miniconda3/envs/ngs/share/snpeff-4.3.1m-0/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[475] that is installed.

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

[473] http://snpeff.sourceforge.net/index.html
[474] http://snpeff.sourceforge.net/index.html
[475] http://snpeff.sourceforge.net/index.html

```
cp /home/manager/miniconda3/envs/ngs/share/snpeff-4.3.1m-0/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 yeast genome
yeastanc.genome : WildYeastAnc
```

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

```
# create folders
mkdir -p ./data/yeastanc
```

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

```
cp ../assembly/spades-final/scaffolds.fasta ./data/yeastanc/sequences.fa
gzip ./data/yeastanc/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/your_new_fungus.gff ./data/yeastanc/genes.gff
gzip ./data/yeastanc/genes.gff
```

Now we can build a new SnpEff[476] database:

```
snpEff build -c snpEff.config -gff3 -v yeastanc > 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/your_new_fungus.gff -o ./data/yeastanc/genes.gtf
gzip ./data/yeastanc/genes.gtf
```

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

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

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

10.6.3 SNP annotation

Now we can use our new SnpEff[477] database to annotate some variants, e.g.:

```
snpEff -c snpEff.config yeastanc ../variants/evolved-6.freebayes.filtered.vcf.gz > evolved-6.
↪freebayes.filtered.anno.vcf
```

SnpEff[478] 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 79).

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[479] will add fields here :

```
# evolved-6.freebayes.filtered.vcf (the original), column 8
AB=0.5;ABP=3.0103;AC=1;AF=0.5;AN=2;AO=56;CIGAR=1X;DP=112;DPB=112;DPRA=0;EPP=3.16541;EPPR=3.16541;
↪GTI=0;LEN=1;MEANALT=1;MQM=42;MQMR=42;NS=1;NUMALT=1;ODDS=331.872;PAIRED=1;PAIREDR=1;PAO=0;PQA=0;
↪PQR=0;PRO=0;QA=2128;QR=2154;RO=56;RPL=35;RPP=10.6105;RPPR=3.63072;RPR=21;RUN=1;SAF=30;SAP=3.63072;
↪SAR=26;SRF=31;SRP=4.40625;SRR=25;TYPE=snp

# evolved-6.freebayes.filtered.anno.vcf, column 8
AB=0.5;ABP=3.0103;AC=1;AF=0.5;AN=2;AO=56;CIGAR=1X;DP=112;DPB=112;DPRA=0;EPP=3.16541;EPPR=3.16541;
↪GTI=0;LEN=1;MEANALT=1;MQM=42;MQMR=42;NS=1;NUMALT=1;ODDS=331.872;PAIRED=1;PAIREDR=1;PAO=0;PQA=0;
↪PQR=0;PRO=0;QA=2128;QR=2154;RO=56;RPL=35;RPP=10.6105;RPPR=3.63072;RPR=21;RUN=1;SAF=30;SAP=3.63072;
↪SAR=26;SRF=31;SRP=4.40625;SRR=25;TYPE=snp;ANN=T|missense_variant|MODERATE|CDS_NODE_40_length_1292_
↪cov_29.5267_1_1292|GENE_CDS_NODE_40_length_1292_cov_29.5267_1_1292|transcript|TRANSCRIPT_CDS_NODE_
↪40_length_1292_cov_29.5267_1_1292|protein_coding|1/1|c.664T>A|p.Ser222Thr|664/1292|664/1292|222/
↪429||WARNING_TRANSCRIPT_INCOMPLETE,T|intragenic_variant|MODIFIER|GENE_NODE_40_length_1292_cov_29.
↪5267_1_1292|GENE_NODE_40_length_1292_cov_29.5267_1_1292|gene_variant|GENE_NODE_40_length_1292_cov_
↪29.5267_1_1292|||n.629A>T||||||
```

When expecting the second entry, we find that SnpEff[480] 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 threonine to a serine (`c.664T>A|p.Ser222Thr`). The codon for serine is `TCN` and for threonine is `ACN`, so the variant in the first nucleotide of the codon made the amino acid change.

A quick protein BLAST[481] of the CDS sequence where the variant was found (extracted from the `genes.gff.gz`) shows that the closest hit is a translation elongation factor from a species called Candida dubliniensis[482] another fungi.

[477] http://snpeff.sourceforge.net/index.html
[478] http://snpeff.sourceforge.net/index.html
[479] http://snpeff.sourceforge.net/index.html
[480] http://snpeff.sourceforge.net/index.html
[481] https://blast.ncbi.nlm.nih.gov/Blast.cgi
[482] https://en.wikipedia.org/wiki/Candida_dubliniensis
[483] https://blast.ncbi.nlm.nih.gov/Blast.cgi

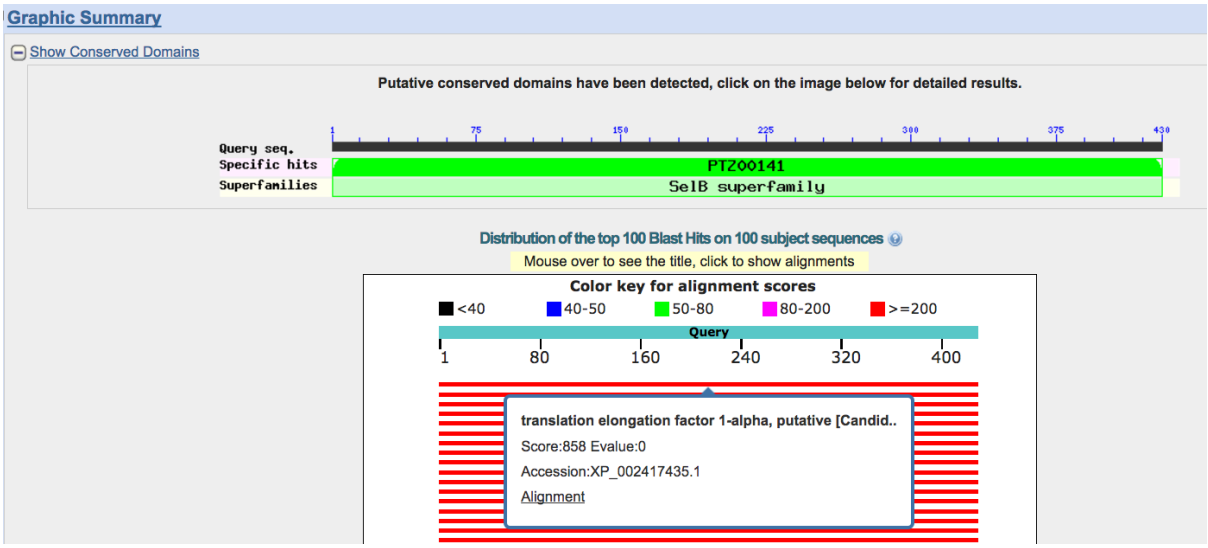


Fig. 10.2: Results of a BLAST[483] search of the CDS.

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: FastQC

Create directory:

```
mkdir trimmed-fastqc
```

Run FastQC:

```
fastqc -o trimmed-fastqc trimmed/ancestor-R1.fastq.trimmed.gz trimmed/ancestor-R2.fastq.trimmed.gz␣
↪trimmed/evolved-6-R1.fastq.trimmed.gz trimmed/evolved-6-R2.fastq.trimmed.gz
```

Open html webpages:

```
firefox trimmed-fastqc/\*.html
```

12.1.2 Code: SolexaQA++ trimming

Create directory for result-files:

```
mkdir trimmed
```

Run SolexaQA++:

```
./SolexaQA++ dynamictrim -p 0.05 -d trimmed/ data/ancestor-R1.fastq.gz

./SolexaQA++ dynamictrim -p 0.05 -d trimmed/ data/ancestor-R2.fastq.gz

./SolexaQA++ dynamictrim -p 0.05 -d trimmed/ data/evolved-6-R1.fastq.gz

./SolexaQA++ dynamictrim -p 0.05 -d trimmed/ data/evolved-6-R2.fastq.gz
```

12.1.3 Code: SolexaQA++ qc

Create directory for result-files:

```
mkdir trimmed-solexaqa/
```

Run SolexaQA++:

```
./SolexaQA++ analysis -d trimmed-solexaqa trimmed/ancestor-R1.fastq.trimmed.gz

./SolexaQA++ analysis -d trimmed-solexaqa trimmed/ancestor-R2.fastq.trimmed.gz

./SolexaQA++ analysis -d trimmed-solexaqa trimmed/evolved-6-R1.fastq.trimmed.gz

./SolexaQA++ analysis -d trimmed-solexaqa trimmed/evolved-6-R2.fastq.trimmed.gz
```

12.2 Assembly

12.2.1 Code: SPAdes assembly (trimmed data)

```
spades.py -o assembly/spades-150/ -k 21,33,55,77 --careful -1 trimmed/ancestor-R1.fastq.trimmed.gz -
↪2 trimmed/ancestor-R2.fastq.trimmed.gz
```

12.2.2 Code: SPAdes assembly (original data)

```
spades.py -o assembly/spades-original/ -k 21,33,55,77 --careful -1 data/ancestor-R1.fastq.gz -2␣
↪data/ancestor-R2.fastq.gz
```

12.3 Mapping

12.3.1 Code: Bowtie2 indexing

Build the index:

```
bowtie2-build assembly/spades-final/scaffolds.fasta assembly/spades-final/scaffolds
```

12.3.2 Code: Bowtie2 mapping

Map to the genome. Use a max fragemnt length of 1000 bp:

```
bowtie2 -X 1000 -x assembly/spades-final/scaffolds -1 trimmed/evolved-6-R1.fastq.trimmed.gz -2␣
↪trimmed/evolved-6-R2.fastq.trimmed.gz -S mappings/evolved-6.sam
```

12.3.3 Code: BWA indexing

Index the genome assembly:

```
bwa index assembly/spades-final/scaffolds.fasta
```

12.3.4 Code: BWA mapping

Run bwa mem:

```
bwa mem assembly/spades-final/scaffolds.fasta trimmed/evolved-6-R1.fastq.trimmed.gz trimmed/evolved-
↪6-R2.fastq.trimmed.gz > mappings/evolved-6.sam
```

CHAPTER

THIRTEEN

DOWNLOADS

13.1 Tools

- Miniconda installer [EXTERNAL[565] | INTERNAL[566] | DROPBOX[567]]
- Minikraken database [EXTERNAL[568] | INTERNAL[569] | DROPBOX[570]]
- Centrifuge database [EXTERNAL[571] | INTERNAL[572] | DROPBOX[573]]
- Krona taxonomy database [INTERNAL[574] | DROPBOX[575]]
- SolexaQA++ [EXTERNAL[576] | INTERNAL[577] | DROPBOX[578]]
- BUSCO Saccharomycetales_odb9 database [EXTERNAL[579] | INTERNAL[580] | DROPBOX[581]]
- IGV [EXTERNAL[582] | INTERNAL[583] | DROPBOX[584]]
- RAxML-NG [EXTERNAL[585] | INTERNAL[586] | DROPBOX[587]]

13.2 Data

- *Quality control* (page 9): Raw data-set [INTERNAL[588] | DROPBOX[589]]
- *Quality control* (page 9): Trimmed data-set [INTERNAL[590] | DROPBOX[591]]

[565] https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
[566] http://compbio.massey.ac.nz/data/203341/Miniconda3-latest-Linux-x86_64.sh
[567] https://www.dropbox.com/s/tz2wocdzjr4grdy/Miniconda3-latest-Linux-x86_64.sh?dl=0
[568] http://ccb.jhu.edu/software/kraken/dl/minikraken.tgz
[569] http://compbio.massey.ac.nz/data/203341/minikraken.tgz
[570] https://www.dropbox.com/s/lje0ykzdxtq3rpk/minikraken.tgz?dl=0
[571] ftp://ftp.ccb.jhu.edu/pub/infphilo/centrifuge/data/p_compressed.tar.gz
[572] http://compbio.massey.ac.nz/data/203341/p_compressed.tar.gz
[573] https://www.dropbox.com/s/a4u6u37ngk2nawx/p_compressed.tar.gz?dl=0
[574] http://compbio.massey.ac.nz/data/203341/taxonomy.tab.gz
[575] https://www.dropbox.com/s/cwf1qc5zyq65yvn/taxonomy.tab.gz?dl=0
[576] https://downloads.sourceforge.net/project/solexaqa/src/SolexaQA%2B%2B_v3.1.7.1.zip?r=https%3A%2F%2Fsourceforge.net%2Fprojects%2Fsolexaqa%2Ffiles%2F&ts=1495062885&use_mirror=iweb
[577] http://compbio.massey.ac.nz/data/203341/SolexaQA.tar.gz
[578] https://www.dropbox.com/s/r9a7hg0tlwe6pk4/SolexaQA.tar.gz?dl=0
[579] http://busco.ezlab.org/datasets/saccharomycetales_odb9.tar.gz
[580] http://compbio.massey.ac.nz/data/203341/saccharomycetales_odb9.tar.gz
[581] https://www.dropbox.com/s/7ow5yi6s5a0ente/saccharomycetales_odb9.tar.gz?dl=0
[582] http://data.broadinstitute.org/igv/projects/downloads/IGV_2.3.92.zip
[583] http://compbio.massey.ac.nz/data/203341/IGV_2.3.92.zip
[584] https://www.dropbox.com/s/bpucaolxhwf78le/IGV_2.3.92.zip?dl=0
[585] https://github.com/amkozlov/raxml-ng/releases/download/0.3.0/raxml-ng_v0.3.0b_linux_x86_64.zip
[586] http://compbio.massey.ac.nz/data/203341/raxml-ng_v0.3.0b_linux_x86_64.zip
[587] https://www.dropbox.com/s/iliws53ri5z4y69/raxml-ng_v0.3.0b_linux_x86_64.zip?dl=0
[588] http://compbio.massey.ac.nz/data/203341/data.tar.gz
[589] https://www.dropbox.com/s/70gcfqzrqugwcn5/data.tar.gz?dl=0
[590] http://compbio.massey.ac.nz/data/203341/trimmed.tar.gz
[591] https://www.dropbox.com/s/o6ioadoxfppbjrv/trimmed.tar.gz?dl=0

- *Genome assembly* (page 23): Assembled data-set [INTERNAL[592] | DROPBOX[593]]
- *Read mapping* (page 29): Mapping index (bowtie2) [INTERNAL[594] | DROPBOX[595]]
- *Read mapping* (page 29): Mapping index (bwa) [INTERNAL[596] | DROPBOX[597]]
- *Read mapping* (page 29): Mapped data [INTERNAL[598] | DROPBOX[599]]
- *Genome annotation* (page 59): GFF annotation file [INTERNAL[600] | DROPBOX[601]]
- *Orthology and Phylogeny* (page 65): *S. cerevisiase* TEF2 gene file [INTERNAL[602] | DROPBOX[603]]
- *Orthology and Phylogeny* (page 65): BLAST file [INTERNAL[604] | DROPBOX[605]]
- *Variants-of-interest* (page 69): SnpEff annotated vcf-file [INTERNAL[606] | DROPBOX[607]]

[592] http://compbio.massey.ac.nz/data/203341/assembly.tar.gz
[593] https://www.dropbox.com/s/vlyn2fxgkiml5m8/assembly.tar.gz?dl=0
[594] http://compbio.massey.ac.nz/data/203341/bowtie2-index.tar.gz
[595] https://www.dropbox.com/s/dcbdsxl5bjhmif8/bowtie2-index.tar.gz?dl=0
[596] http://compbio.massey.ac.nz/data/203341/bwa-index.tar.gz
[597] https://www.dropbox.com/s/yidw27u56iast9z/bwa-index.tar.gz?dl=0
[598] http://compbio.massey.ac.nz/data/203341/evolved-6.sorted.dedup.bam
[599] https://www.dropbox.com/s/k1qn63rwnojhmrz/evolved-6.sorted.dedup.bam?dl=0
[600] http://compbio.massey.ac.nz/data/203341/your_new_fungus.gff.gz
[601] https://www.dropbox.com/s/6bo9g8h3q6h1x8x/your_new_fungus.gff.gz?dl=0
[602] http://compbio.massey.ac.nz/data/203341/s_cerev_tef2.fas
[603] https://www.dropbox.com/s/ooxl2q0vp0bzmq3/s_cerev_tef2.fas?dl=0
[604] http://compbio.massey.ac.nz/data/203341/blast_u.fas
[605] https://www.dropbox.com/s/ul4dzx44lfoewzx/blast_u.fas?dl=0
[606] http://compbio.massey.ac.nz/data/203341/evolved-6.freebayes.filtered.anno.vcf
[607] https://www.dropbox.com/s/67m45v5fghdh0d3/evolved-6.freebayes.filtered.anno.vcf?dl=0

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]

[COX2010] Cox MP, Peterson DA and Biggs PJ. SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. BMC Bioinformatics, 2010, 11:485. DOI: 10.1186/1471-2105-11-485[110]

[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[111]

[KIRCHNER2014] Kirchner et al. Addressing challenges in the production and analysis of Illumina sequencing data. BMC Genomics (2011) 12:382[112]

[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[113]

[ROBASKY2014] Robasky et al. The role of replicates for error mitigation in next-generation sequencing. Nature Reviews Genetics (2014) 15, 56-62[114]

[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.[167] 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[168]

[GUREVICH2013] Gurevich A, Saveliev V, Vyahhi N and Tesler G. QUAST: quality assessment tool for genome assemblies. Bioinformatics 2013, 29(8), 1072-1075[169]

[NAGARAJAN2013] Nagarajan N, Pop M. Sequence assembly demystified. Nat Rev Genet. 2013 Mar;14(3):157-67[170]

[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[171]

[5] http://dx.doi.org/10.1016/j.tree.2012.06.001
[6] http://doi.org/10.1111/j.1567-1364.2006.00061.x
[110] http://doi.org/10.1186/1471-2105-11-485
[111] http://doi.org/10.1111/j.1755-0998.2011.03024.x
[112] http://doi.org/10.1186/1471-2164-12-382
[113] http://doi.org/10.1186/1944-3277-10-18
[114] http://doi.org/10.1038/nrg3655
[167] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4290589/
[168] http://dx.doi.org/10.1038/nbt.2023
[169] http://bioinformatics.oxfordjournals.org/content/29/8/1072
[170] http://dx.doi.org/10.1038/nrg3367
[171] http://genome.cshlp.org/content/22/3/557.full?sid=59ea80f7-b408-4a38-9888-3737bc670876

[WICK2015] Wick RR, Schultz MB, Zobel J and Holt KE. Bandage: interactive visualization of de novo genome assemblies. Bioinformatics 2015, 10.1093/bioinformatics/btv383[172]

[TRAPNELL2009] Trapnell C, Salzberg SL. How to map billions of short reads onto genomes. Nat Biotechnol. (2009) 27(5):455-7. doi: 10.1038/nbt0509-455.[247]

[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[318]

[ONDOV2011] Ondov BD, Bergman NH, and Phillippy AM. Interactive metagenomic visualization in a Web browser. BMC Bioinformatics, 2011, 12(1):385.[319]

[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[320].

[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[359]

[OLSEN2015] Olsen ND et al. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. Front. Genet., 2015, 6:235.[360]

[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[401]

[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.[402]

[172] http://bioinformatics.oxfordjournals.org/content/early/2015/07/11/bioinformatics.btv383.long
[247] http://doi.org/10.1038/nbt0509-455
[318] https://www.ncbi.nlm.nih.gov/pubmed/27852649
[319] http://www.ncbi.nlm.nih.gov/pubmed/21961884
[320] http://doi.org/10.1186/gb-2014-15-3-r46
[359] http://doi.org/10.1038/nrg2986
[360] https://doi.org/10.3389/fgene.2015.00235
[401] http://doi.org/10.1093/bioinformatics/btv351
[402] https://dx.doi.org/10.1093/nar/gki458