Genomics Tutorial Documentation

Release 2017.1

Sebastian Schmeier

Jan 30, 2017

This is an introductory tutorial for learning genomics mostly on the command-line. You will learn how to analyse next-generation sequencing (NGS) data. The data you will be using is actual research data. The final aim is to identify the genome variations in evolved lines of wild yeast that can explain the observed biological phenotypes.

To this end, you will learn to:

- Make use of a UNIX-based computer environment
- Check the data quality of an NGS experiment
- Create a genome assembly of the ancestor based on NGS data
- Annotate a newly derived genome
- Map NGS reads of evolved lines to the ancestral reference genome
- Call genome variations/mutations in the evolved lines
- Identify the genes responsible for the observed evolved phenotypes

A printable PDF version of this tutorial can be downloaded here.

CHAPTER

ONE

CONTENTS

1.1 Command-line interface introduction

1.1.1 Preface

This tutorial is based on a Linux/Unix *command-line*. Using the *command-line* requires a Linux/Unix operating system. The easiest way to try out a Linux system without actually installing it on your computer is a LiveCD. A LiveCD is a CD/DVD that you prepare (e.g. burn a Linux distribution on it) and insert in your computer. You would restart you computer and can run Linux from the CD/DVD without any installation requirements. This is helpful for trying out a distribution of Linux, not for actual work.

Another route would be to use a virtual machine. A virtual computer that runs within your normal host system, e.g. Windows or MacOSX. The software to create a virtual machine is free, e.g. VirtualBox.

Common flavors of Linux ready for download are e.g. Ubuntu or if you are thinking of going the bioinformatics route, BioLinux, which includes many pre-installed bioinformatics tools (this is also the distribution we will be using).

An accompanying lecture for this tutorial is available at figshare (http://dx.doi.org/10.6084/m9.figshare.1506799).

1.1.2 Learning outcomes

1. Be able to operate comfortably the command-line.
2. Be able to navigate the unix directory structure on the command-line.
3. Be able to start command-line programs and getting help/information about programs.
4. Be able to investigate text files with command-line commands.
5. Be able to investigate the content of text-files on the command-line.
6. Be able to explain the concept of a unix pipe.

1.1.3 Introduction

We will go through some introductory material explaining the shell syntax. This is bash syntax but most of it will work on other shells (tcsh, zsh) as well.

What is a shell? Here I shamelessly quote Wikipedia:

> "In computing, a shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (**CLI**) or graphical user interface (GUI), depending on a computer's role and particular operation..."

> "**CLI** shells allow some operations to be performed faster in some situations, especially when a proper GUI has not been or cannot be created. However, they require the user to memorize all commands and their calling syntax, and also to learn the shell-specific scripting language, for example bash script."

1.1.4 The Ubuntu Linux desktop environment

The default environment in Ubuntu is called Unity and is similar to other user interfaces found in Windows or MacOSX (Fig. 1.1).

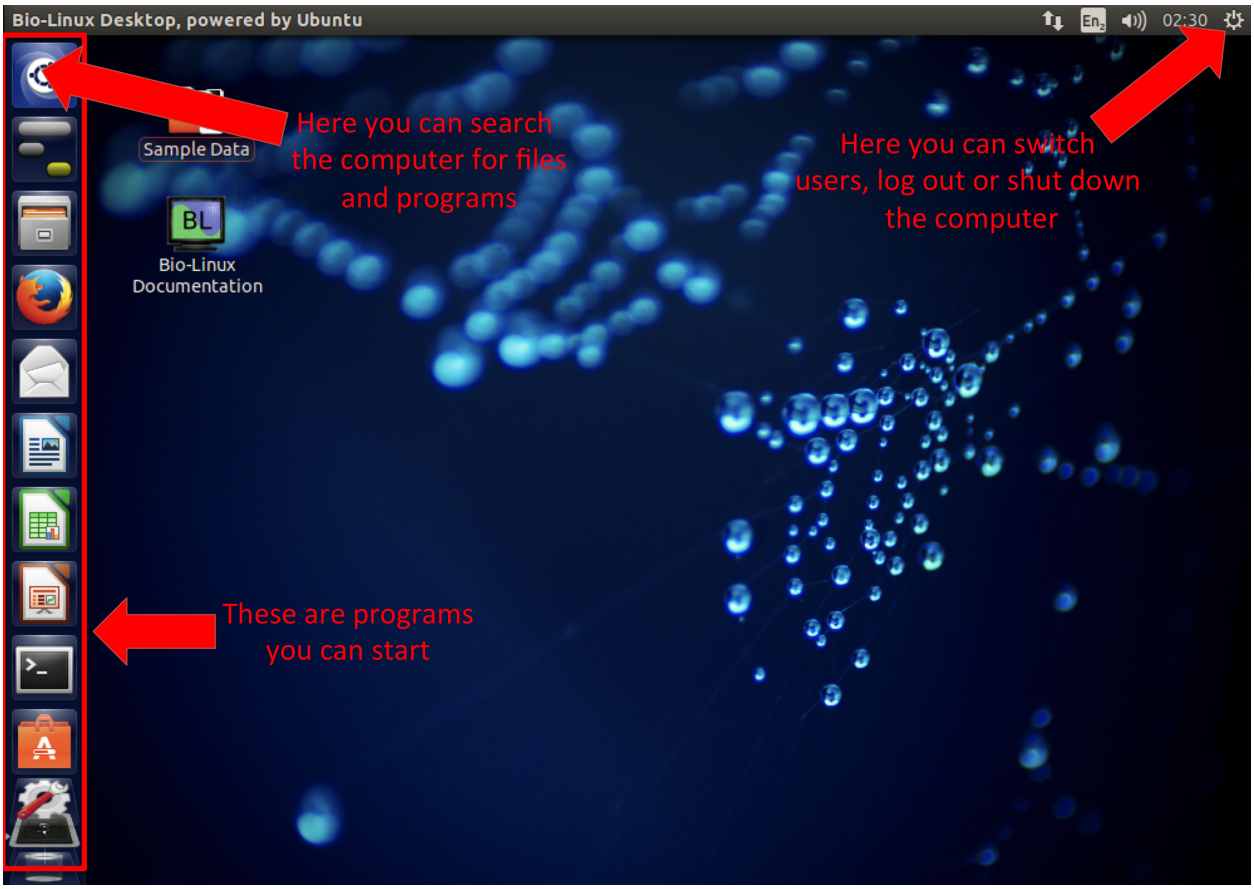

Fig. 1.1: The BioLinux desktop environment Unity.

1.1.5 Some words regarding the Linux file-system

The directory structure in a Linux system is not much different from any other system you worked with, e.g. Windows, MacOSX. It is essentially a tree structure (Fig. 1.2).

To navigate the file-system you can use a file-manager e.g. "Files" the default file manager in the Unity window manager used by BioLinux (Fig. 1.3).

However, on the command-line we navigate via commands and not via mouse clicks. Why is it necessary to use the command-line in the first place? Strictly speaking it is not, if you do not want to make use of programs on the command-line. However, the power of the Linux system becomes only obvious once we learn to make use of the command-line, thus navigating the directory structure via commands is one of the **most important skills** for you to learn.

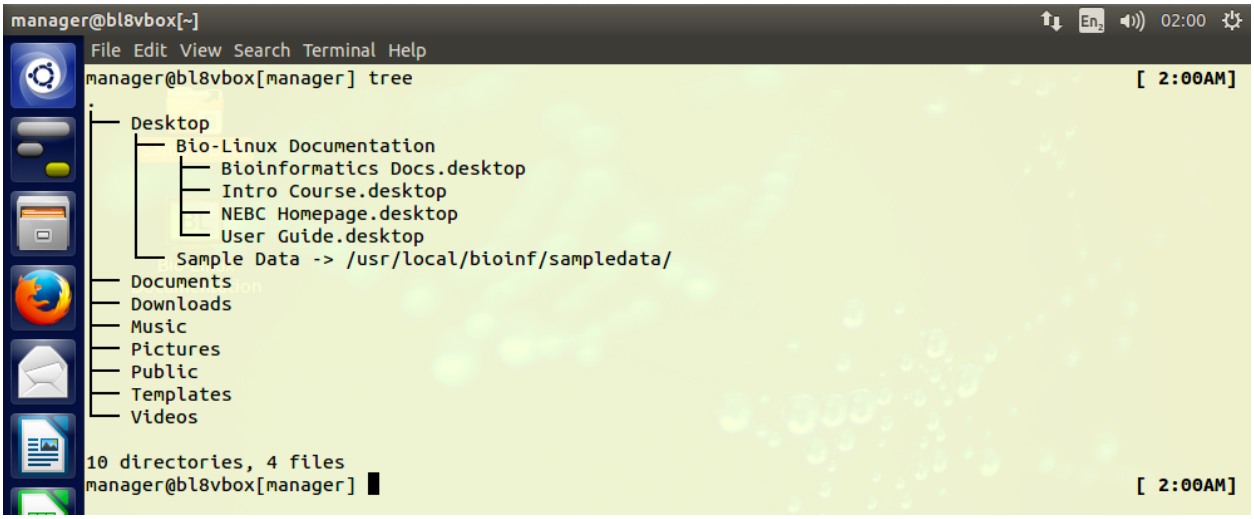

Fig. 1.2: Quick look at the directory tree structure on the command-line.

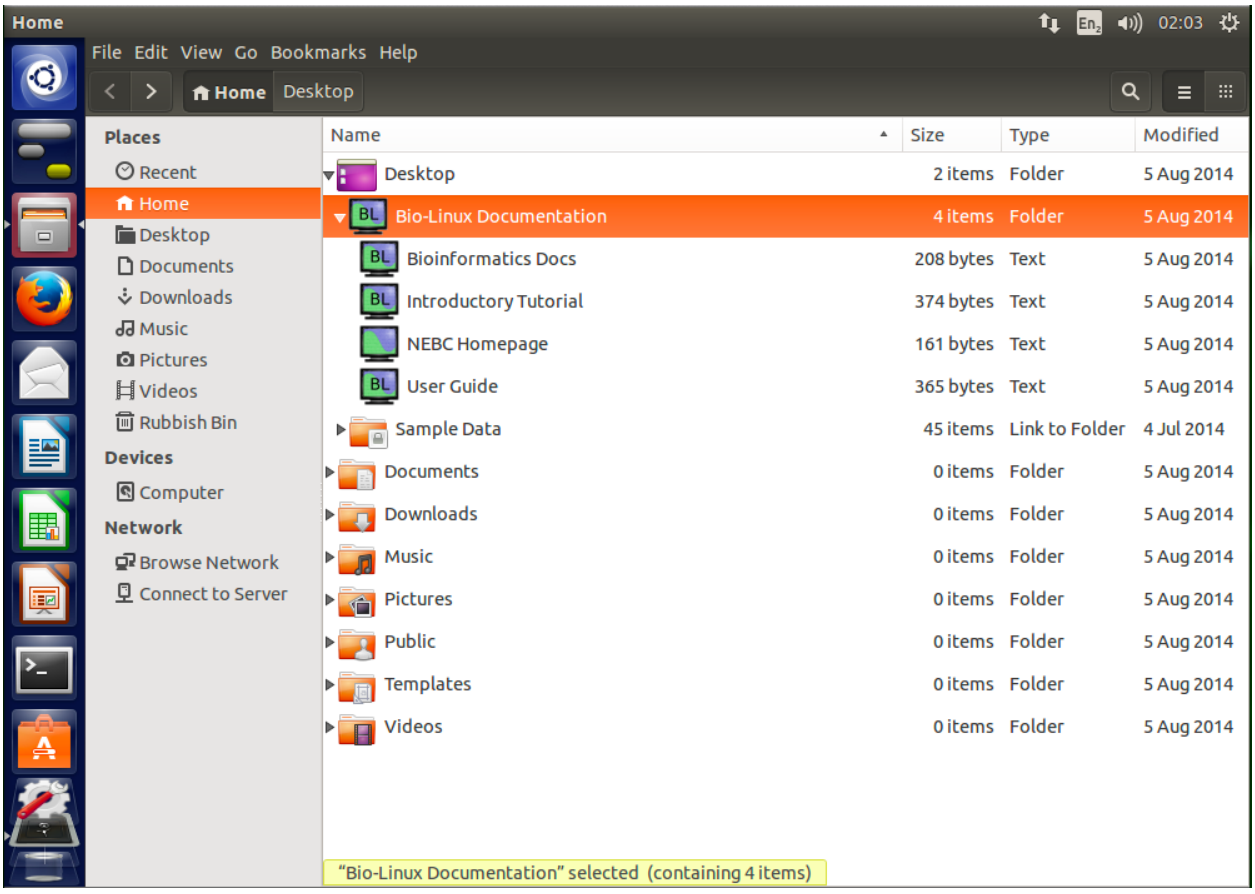

Fig. 1.3: Quick look at the directory tree structure in the "Files" GUI.

1.1.6 Open a terminal

Open a terminal window and you are are ready to go. On your linux desktop find: **Application** –> **Accessories** –> **Terminal** (for Gnome environment) or type "Terminal" in the search box (Fig. 1.4).

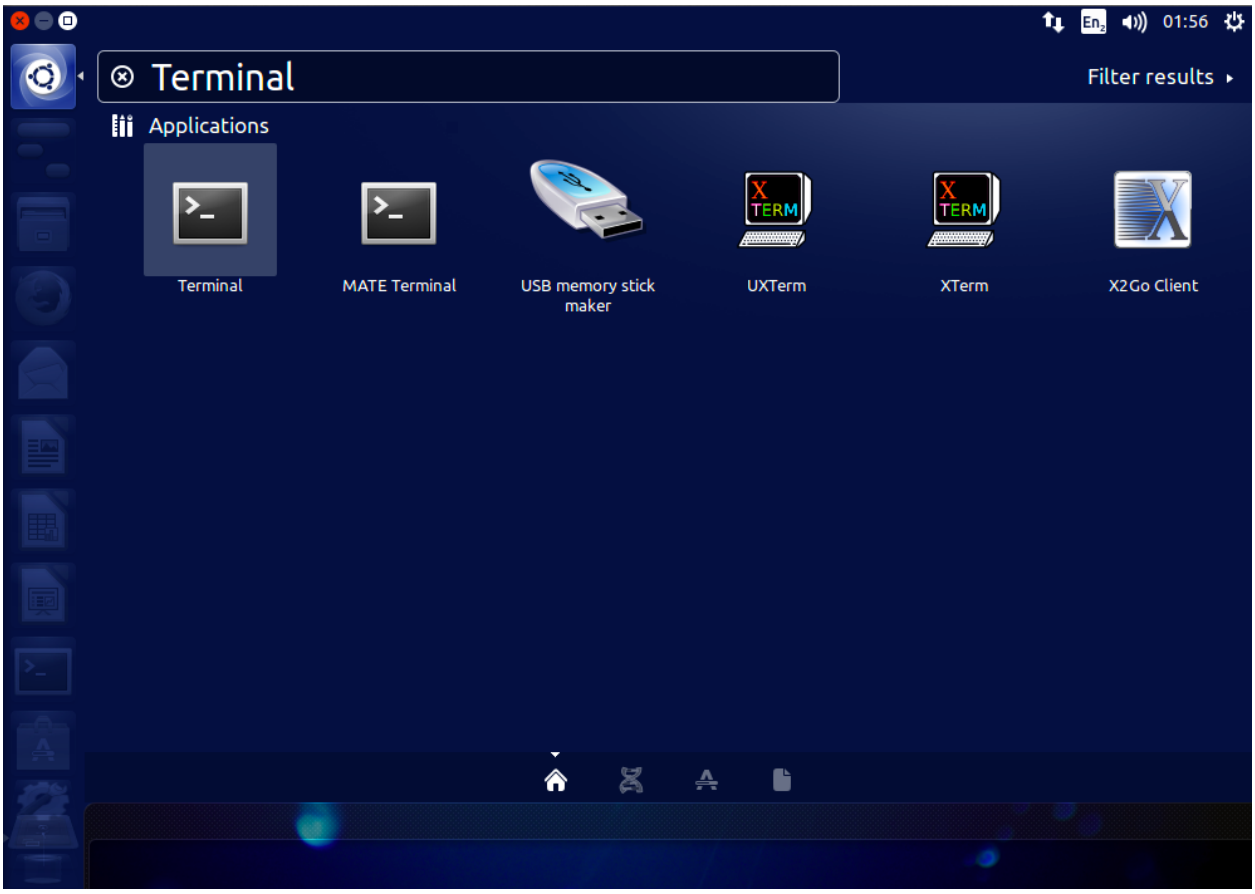

Fig. 1.4: Unity search bar.

Fig. 1.5 shows an example of how a terminal window might look like (it is very easy to change its appearance). You will se this window to execute the commands to work with files and biological data. However, it is by no means restricted to "biological data", once you know how to handle the command-line many tasks based on files will be easily achieved using various programs available here.

Note: The command-line prompt (e.g. $ or >) indicates that the shell/terminal is waiting for commands from us. These will be sent to the computer to execute. As long as you do not see the prompt, the computer is busy processing your request.

1.1.7 Proxy settings

You might encounter problems connecting to the internet. So this is most likely the case if your university has restrictions in place to make the network more secure. One of these measures to make a network more secure is a proxy. However, we need the internet. Follow these steps to get connected.

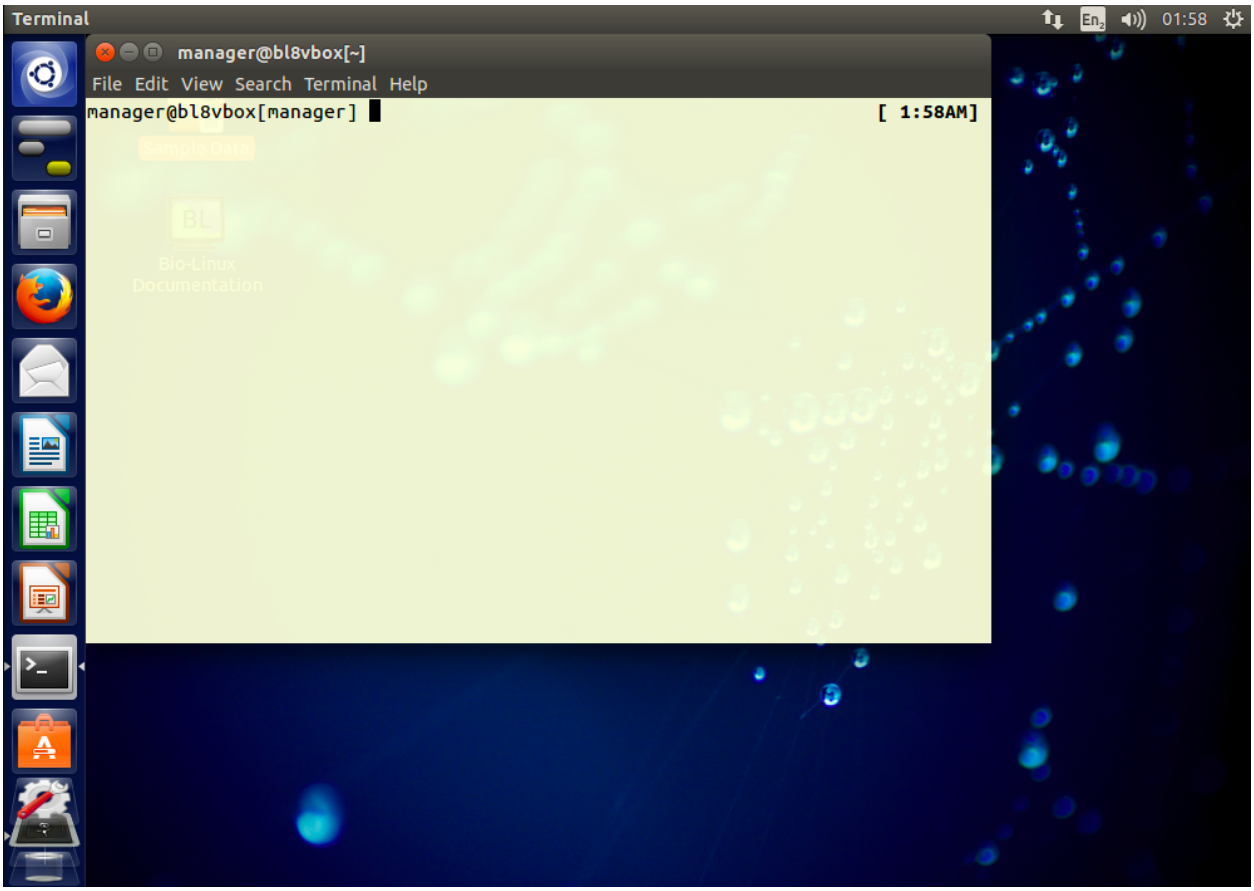

Fig. 1.5: An example of a terminal window in Unity.

Open system settings

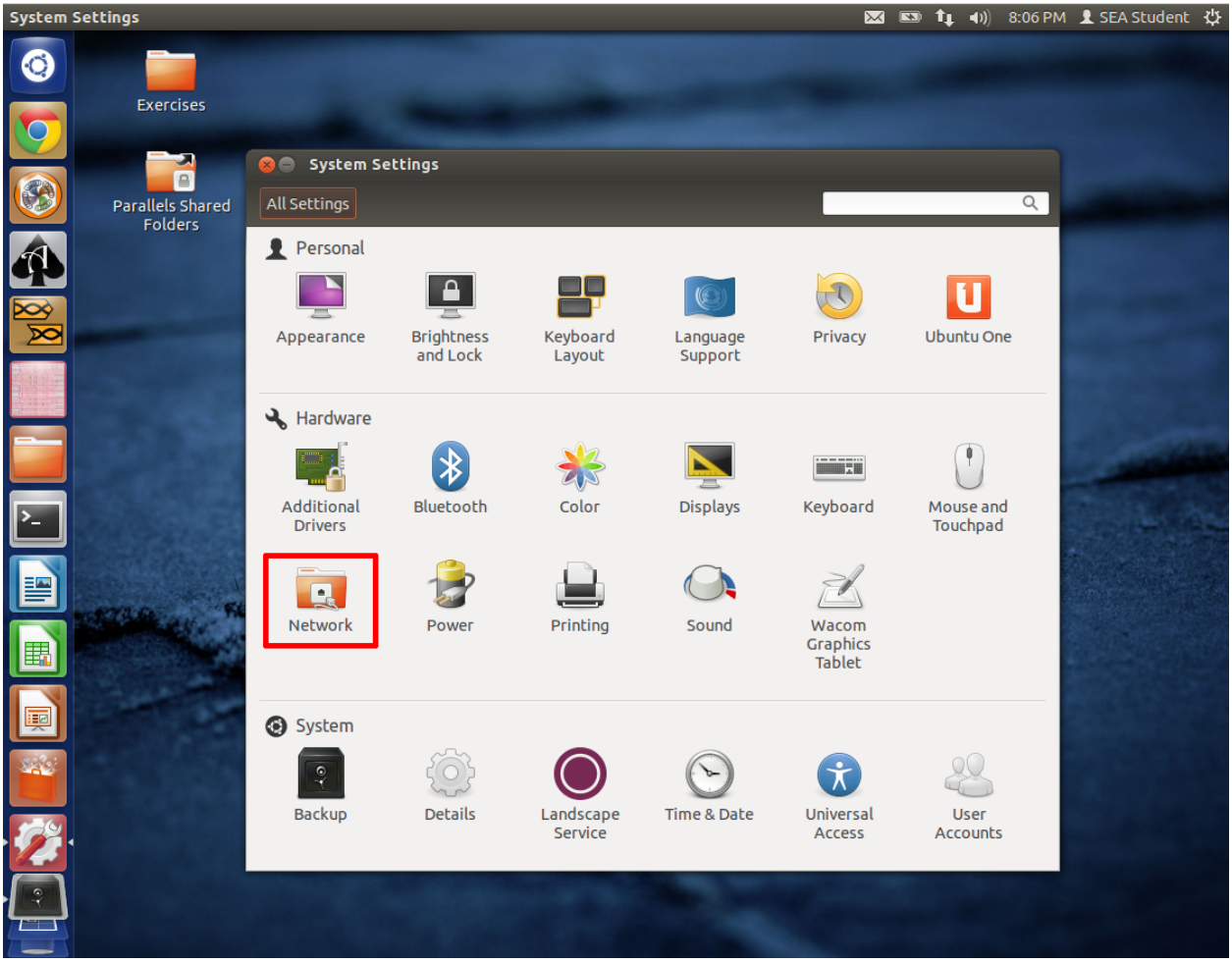

Fig. 1.6: Accessing the system settings.

Open network settings

Change proxy settings to automatic

After changing the proxy settings to automatic you can open a web-browser and you should be asked for you network *username* and *password*. After you typed those and hit *Enter*, you should be connected.

1.1.8 Let's get started

Pre-2016 you would find my own introductory course in this place. However, since 2016 we will be using the excellent material from the Software Carpentry Foundation. I am a SWC affiliated volunteer instructor and we are teaching basic computer skills to scientists, with the goal of general computational up-skilling in the sciences. The material is created in a collaborative manner and tested over and over in many workshops.

Please follow the link to the material at the Software Carpentry (http://swcarpentry.github.io/shell-novice) and have the webpage open. Otherwise you will work in the terminal window aka the shell.

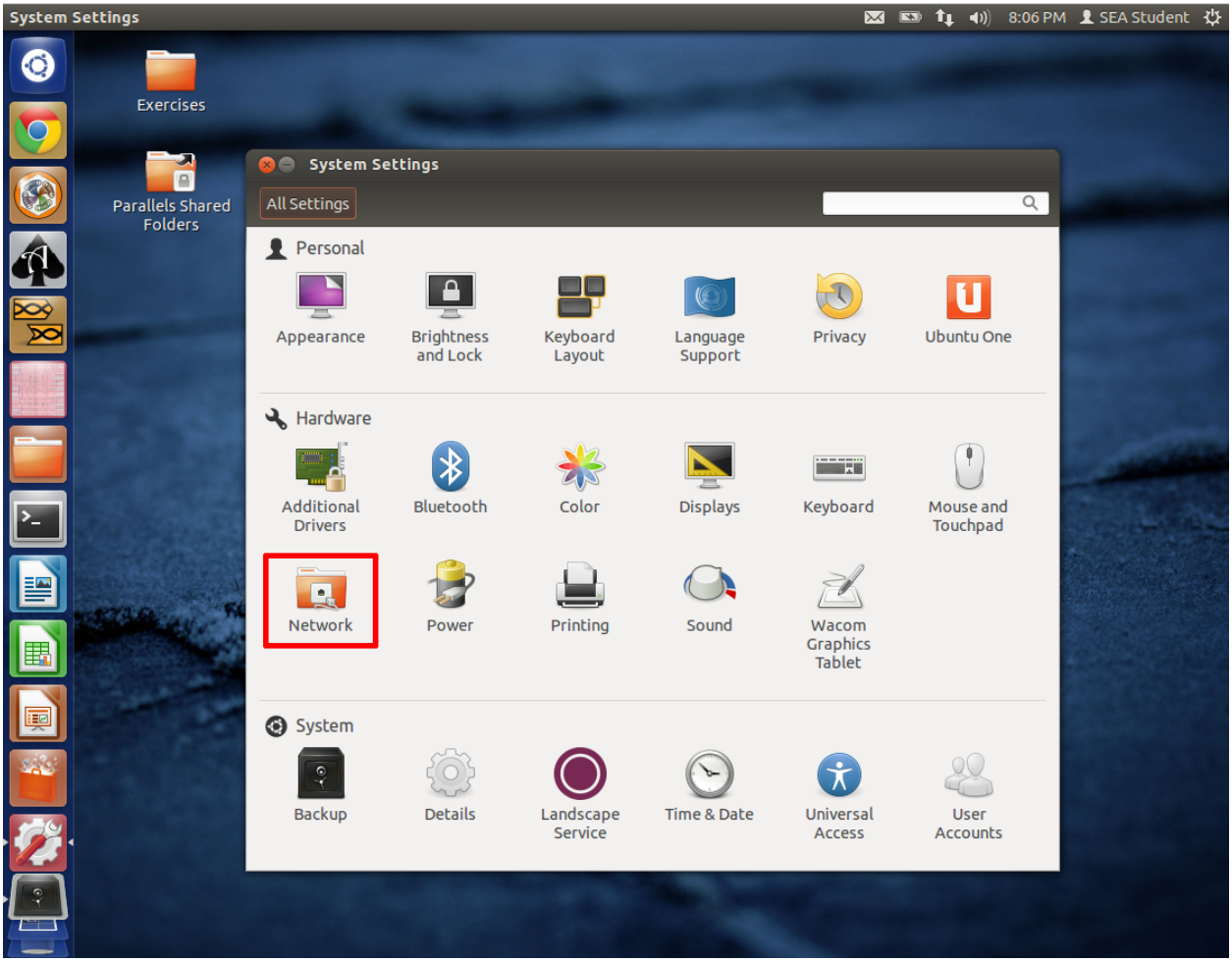

Fig. 1.7: Accessing network settings.

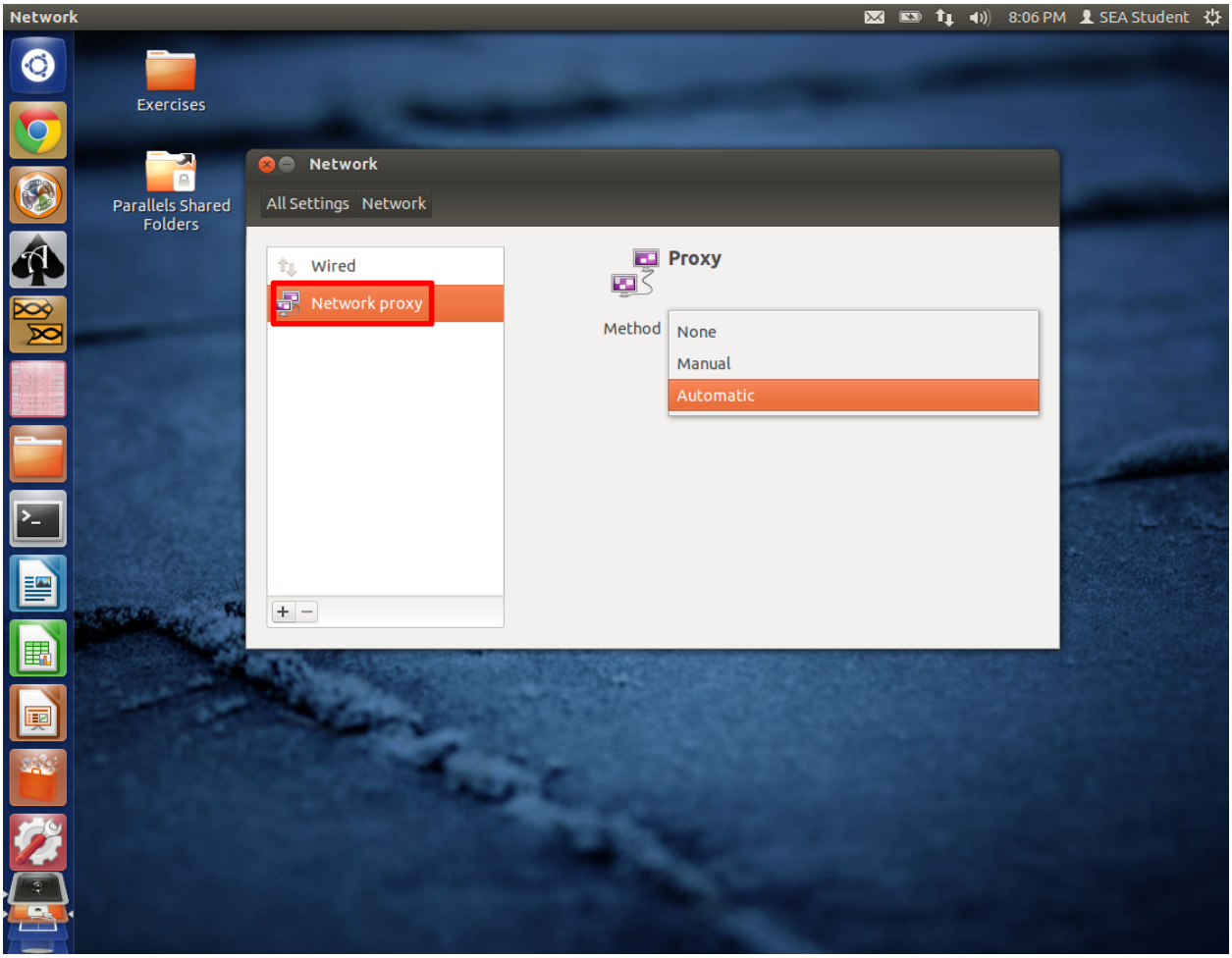

Fig. 1.8: Changing proxy settings.

1.2 NGS - Tool installation

1.2.1 Install the conda package manager

We will use the package/tool managing system conda 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/Miniconda2-latest-Linux-x86_64.sh

# run the installer
bash Miniconda2-latest-MacOSX-x86_64.sh

# delete the installer after successful run
rm Miniconda2-latest-MacOSX-x86_64.sh

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

Close shell/terminal, **re-open** new shell/terminal.

```
conda update conda
```

Attention: Should the conda installer download fail. Please find links to alternative locations on the *Downloads* page.

1.2.2 Create environment

We create a conda 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=2
# activate the environment
source activate ngs
```

1.2.3 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 `source activate ngs` before installing the tools.

1.2.4 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
source activate [name]

# deavtivate env
source deactivate
```

1.3 NGS - Quality control

1.3.1 Preface

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.

There is an accompanying lectures for this tutorial:

- Next-generation sequencing and quality control: An introduction is available at figshare (https://dx.doi.org/10.6084/m9.figshare.2972320.v1).

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

1.3.2 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.

1.3.3 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
```

The data is from a paired-end sequencing run data (see Fig. 1.9), thus we have two files, one for each end of the read.

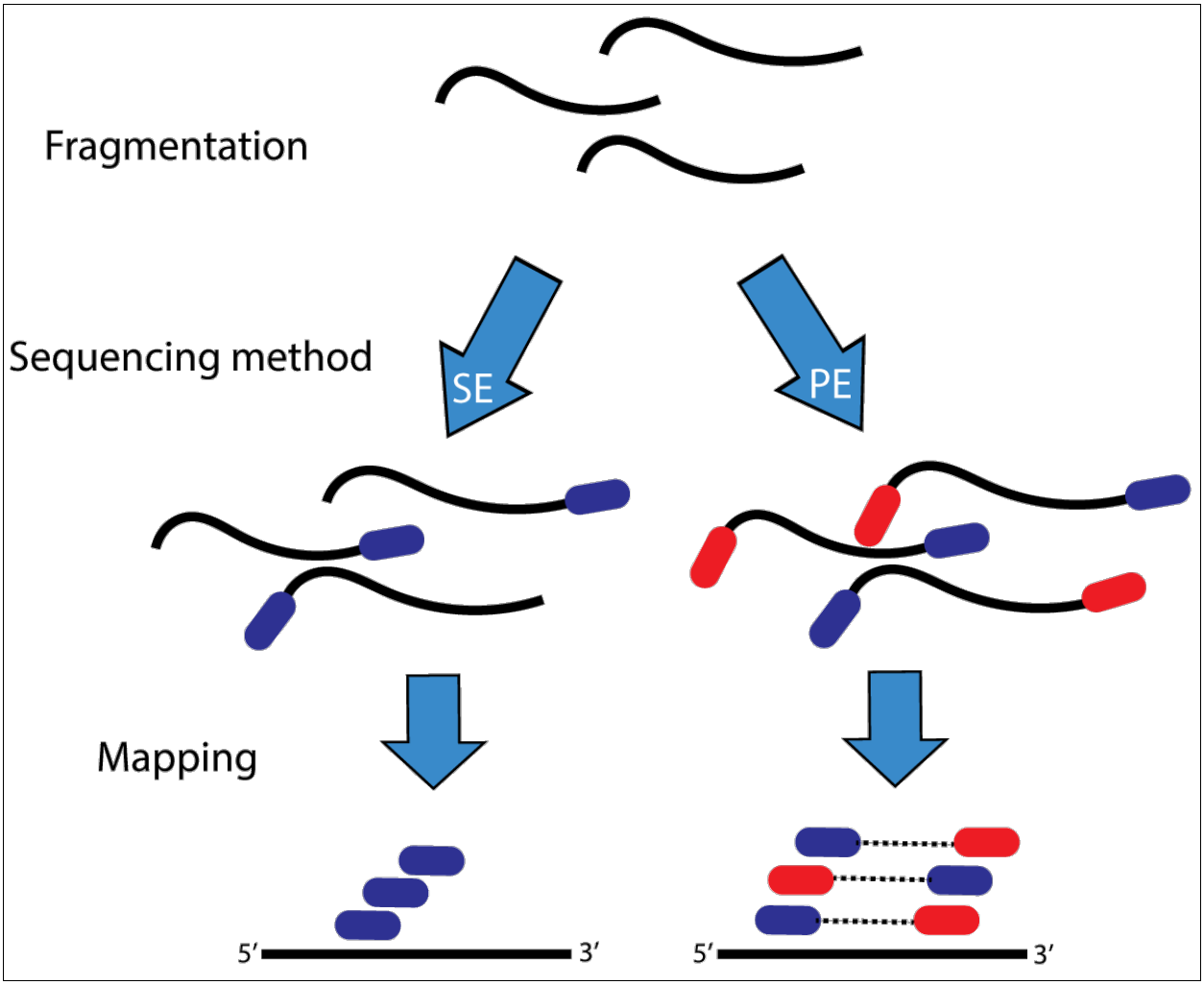

Fig. 1.9: Illustration of single-end (SE) versus paired-end (PE) sequencing.

If you need to refresh how Illumina paired-end sequencing works have a look at the Illumina webpage and this video.

Note: The data we are using is "almost" raw data coming 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 adapters have been removed as well already! The process is explained below but we are not going to do it.

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?

1.3.4 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!

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

Todo

Explain briefly what the quality value represents.

1.3.5 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

1.3.6 PhiX genome

PhiX 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. PhiX is often added at a low known concentration, spiked in the same lane along with the sample or used as a separate lane. As the concentration of the genome is known, one can calibrate the instruments. Thus, PhiX genomic sequences need to be removed before processing your data further as this constitutes a deliberate contamination *[MUKHERJEE2015]*. The steps involve mapping all reads to the "known" PhiX genome, and removing all of those sequence reads from the data.

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.

Note: We are not going to do this step here, as this has been already done. Please see the *NGS - Read mapping* section on how to map reads against a reference genome.

1.3.7 Adapter trimming

The process of sequencing DNA via Illumina 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.

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

Install a tool called fastq-mcf from the ea-utils suite of tools that is able to do this.

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

Todo

Write this section.

1.3.8 Quality assessment of sequencing reads (SolexaQA++)

To assess the sequence read quality of the Illumina run we make use of a program called SolexaQA++ *[COX2010]*. SolexaQA++ was originally developed to work with Solexa data (since bought by Illumina), but long since working with Illumina 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:

> "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."

Install SolexaQA++

Unfortunately, currently we cannot install SolexaQA++ with conda.

```
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*.

SolexaQA++ manual

SolexaQA++ 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, 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.

SolexaQA++ dynamic trimming

We will use SolexaQA++ 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++ `dynamictrim` with the untrimmed data and a probability cutoff of 0.01., 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++ `dynamictrim` produces a graphical output. Explain what the graph shows. Find heklp on the SolexaQA++ website.

Hint: Should you not get 1 and/or 2 right, try these commands here.

SolexaQA++ analysis on trimmed data

Todo

1. Create a directory for the result-files –> **trimmed-solexaqa**.
2. use SolexaQA++ to do the quality assessment with the trimmed data-set.
3. Compare your results to the examples of a particularly bad MiSeq run (Fig. 1.13 to Fig. 1.13, taken from SolexaQA++ website). Write down your observations.
4. What elements in these example figures (Fig. 1.10 to Fig. 1.13) indicate that the show a bad run? Write down your explanations.

Hint: Should you not get 1 and/or 2 it right, try these commands here

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

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

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

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

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 or SolexaQA++ at all to trim the data. You can download the trimmed dataset here. Unarchive and uncompress the files with `tar -xvzf trimmed.tar.gz`.

1.3.10 Quality assessment of sequencing reads (FastQC)

Installing FastQC

```
source activate ngs
conda install fastqc

# should now run the program
fastqc --help
```

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

SYNOPSIS
```

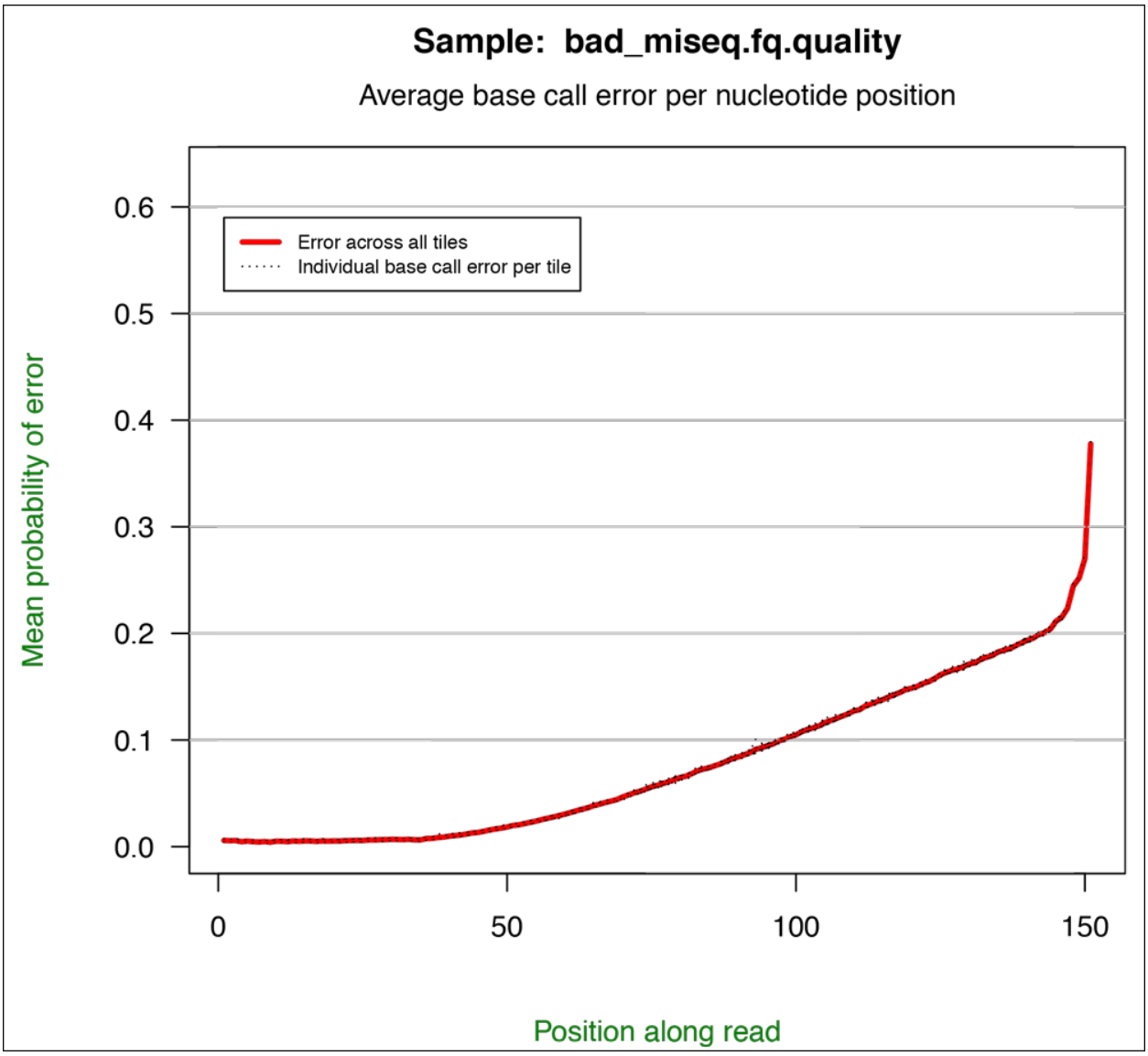

Fig. 1.10: SolexaQA++ example quality plot along reads of a bad MiSeq run

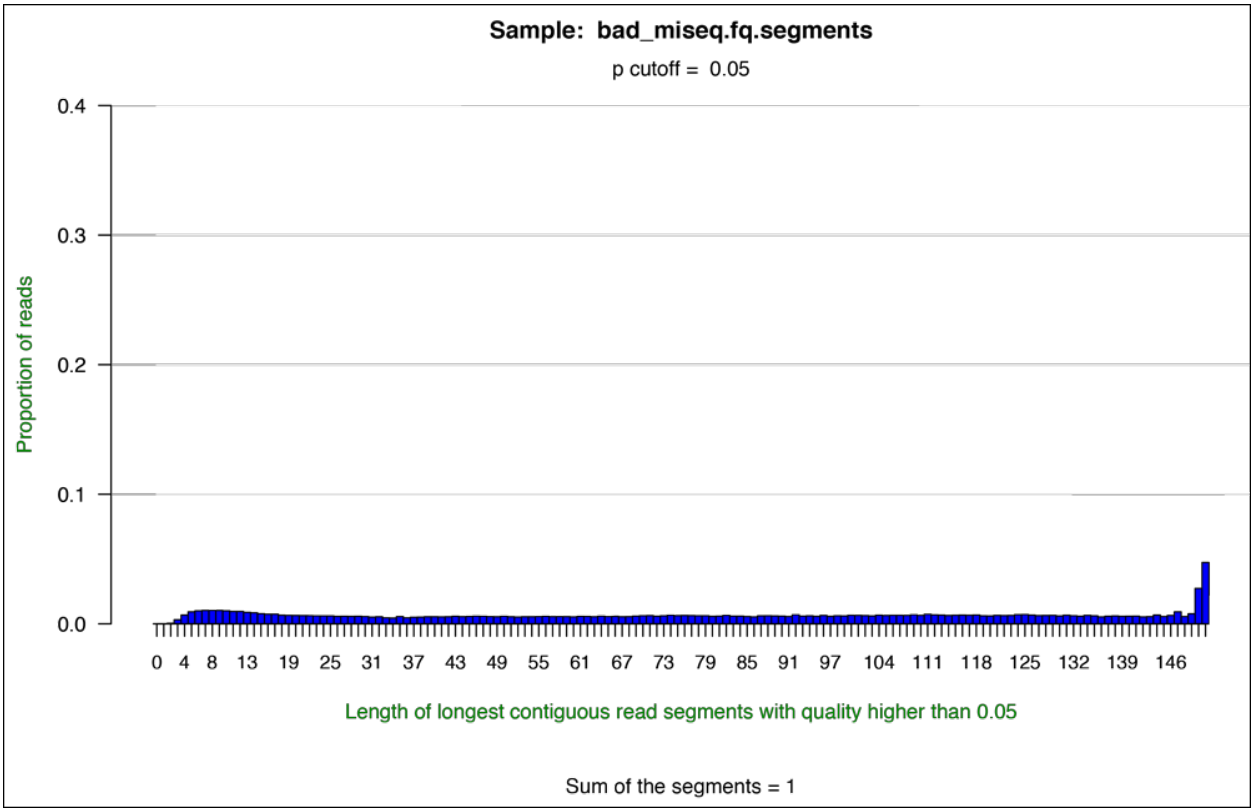

Fig. 1.11: SolexaQA++ example histogram plot of a bad MiSeq run.

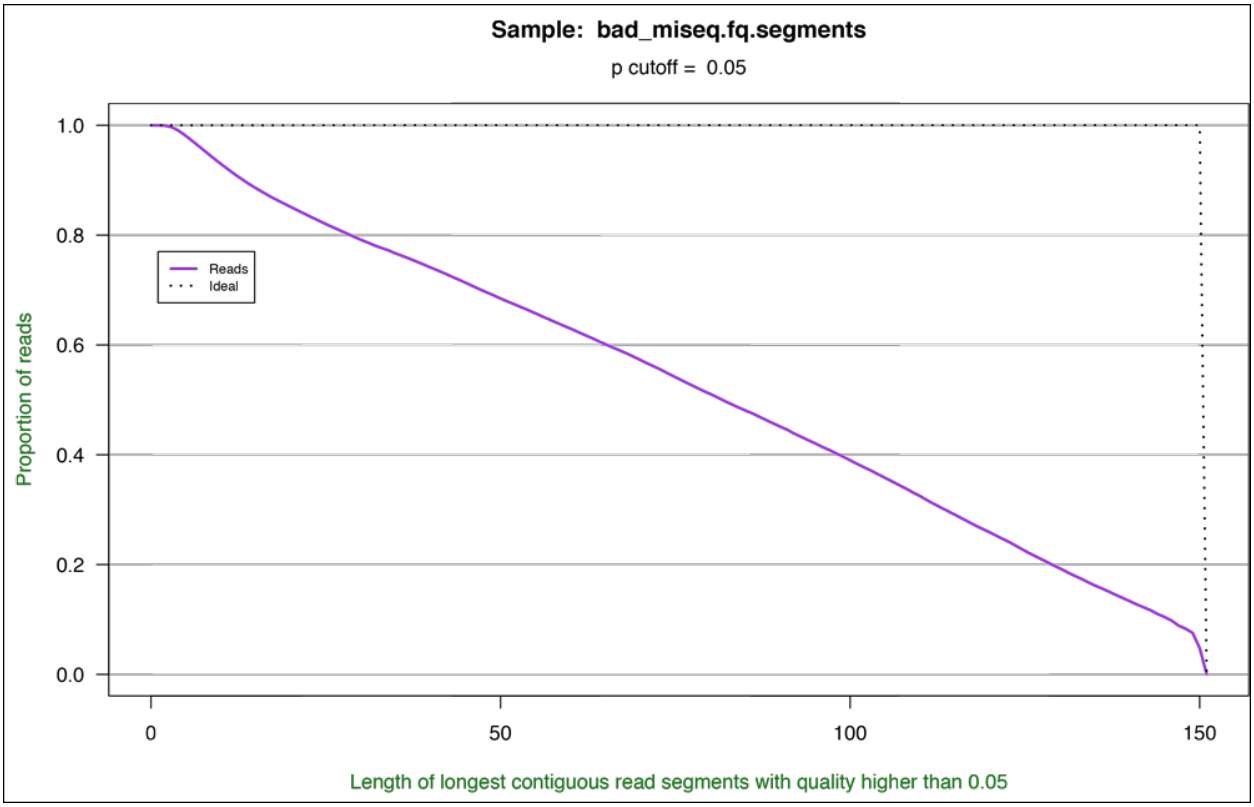

Fig. 1.12: SolexaQA++ example cumulative plot of a bad MiSeq run.

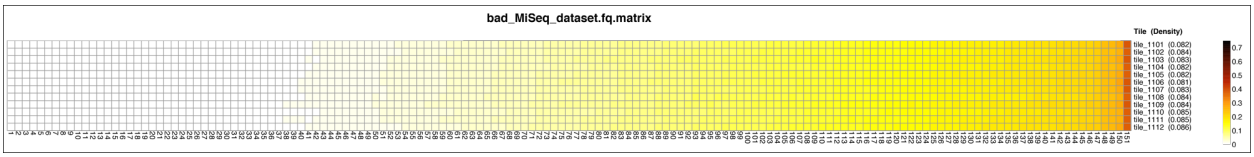

Fig. 1.13: SolexaQA++ example quality heatmap of a bad MiSeq run.

```
        fastqc seqfile1 seqfile2 .. seqfileN

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

DESCRIPTION

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

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

FastQC manual

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

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.

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 website and read about sequencing QC reports for good and bad Illumina sequencing runs.
4. Compare your results to these examples (Fig. 1.14 to Fig. 1.16) of a particularly bad run (taken from the FastQC website) and write down your observations with regards to your data.
5. What elements in these example figures (Fig. 1.14 to Fig. 1.16) indicate that the example is from a bad run?

Hint: Should you not get it right, try these commands here.

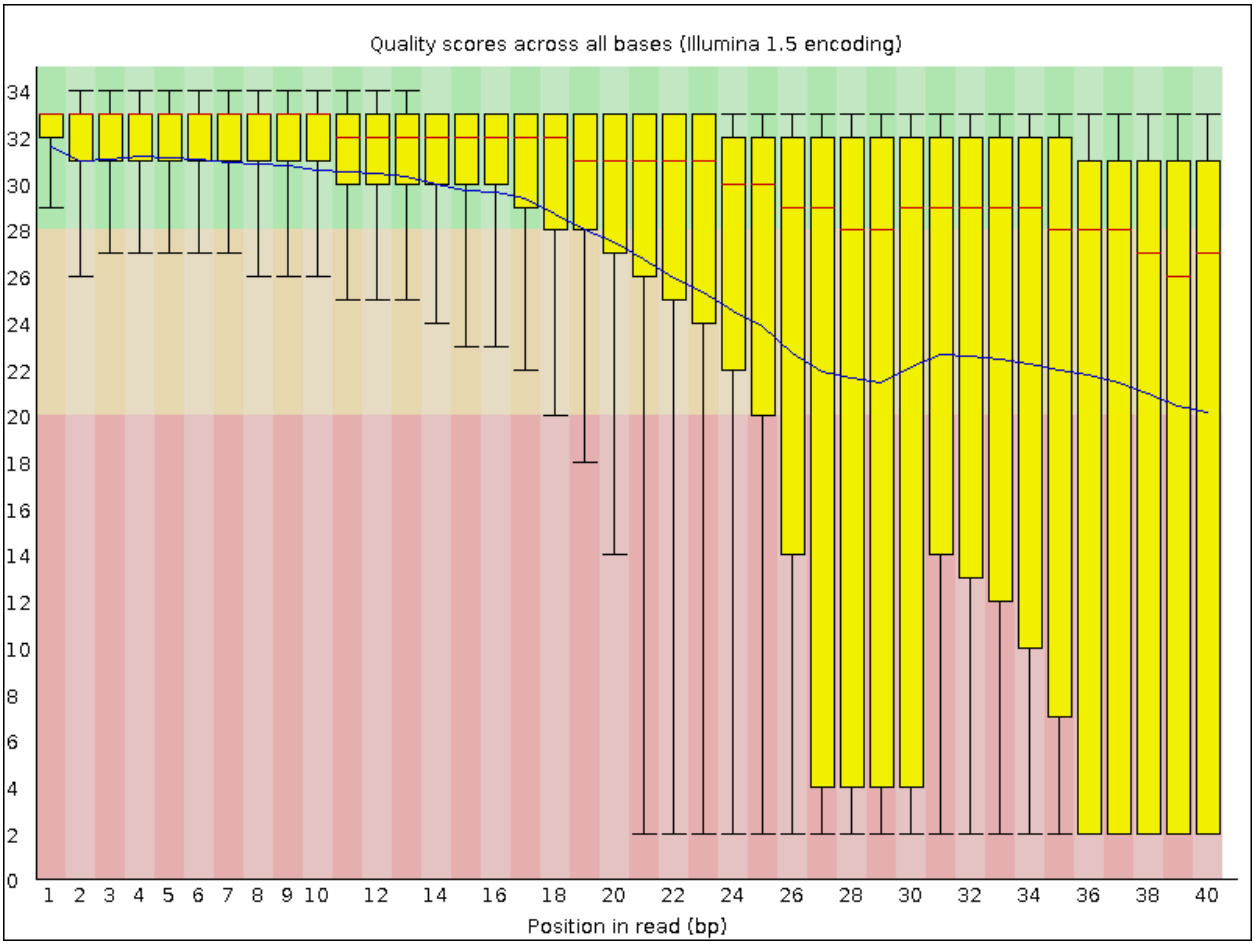

Fig. 1.14: Quality score across bases.

1.3.11 Web links

- Lectures for this topic: Next-generation sequencing and quality control: An introduction
- Illumina paired-end sequencing
- FastQC
- SolexaQA++

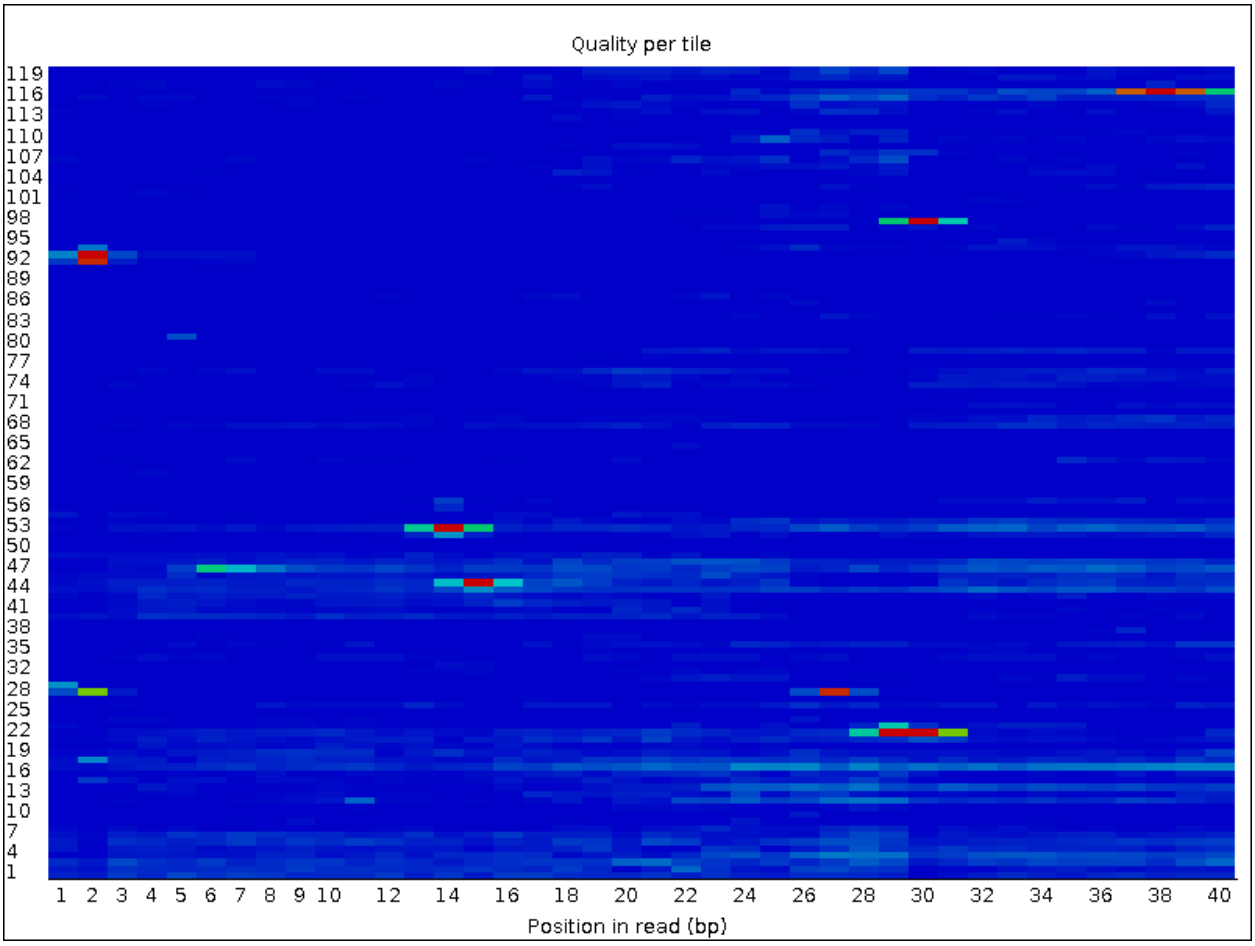

Fig. 1.15: Quality per tile.

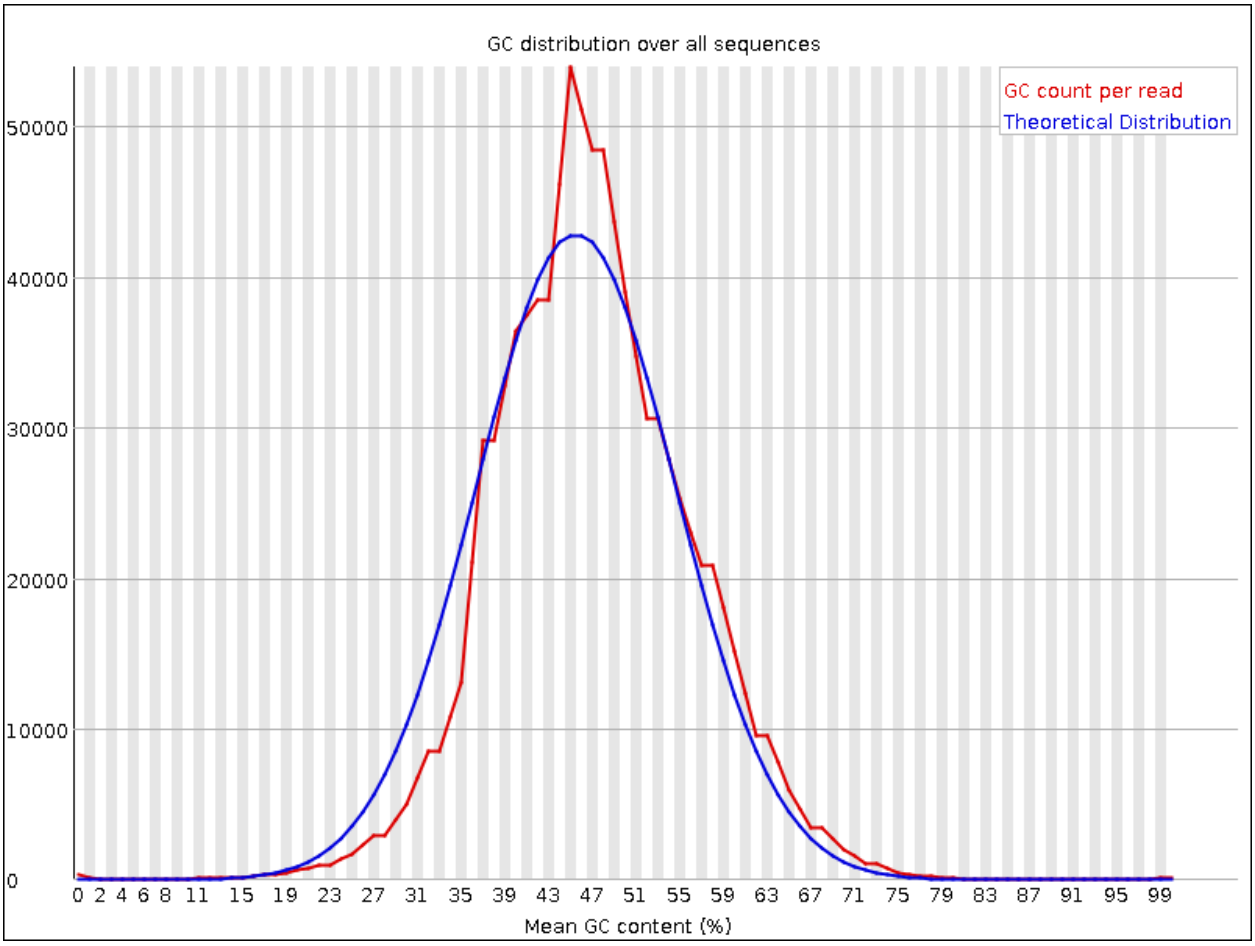

Fig. 1.16: GC distribution over all sequences.

1.4 NGS - Taxonomic investigation

1.4.1 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. However, lets investigate if we find sequences from other species in our sequence set. Maybe they got into the sample through contamination.

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

1.4.2 Kraken

We will be using a tool called Kraken *[WOOD2014]*. This tool uses k-mers to assign a taxonomic labels in form of NCBI Taxonomy 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.

1.4.3 Before we start

Lets see how our directory structure looks so far:

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

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

1.4.4 Installation

Use conda in the same fashion as before to install Kraken:

```
source 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 database that can be used to assign the taxonomic labels to sequences. We opt for downloading the pre-build "minikraken" database from the Kraken 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.

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?

1.4.5 Usage

Now that we have installed Kraken 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 the one it should contain. We call the Kraken 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 compressed fastq-files, so we need to use two additional option switches, `--gzip-compressed` which allows us to input compressed files and `--fastq-input` which tells Kraken that it is dealing with fastq-formated files. Here, we are investigating one of the Paired-end read files of the ancestor.

kraken –only-classified-output –db minikraken_20141208 –gzip-compressed –fastq-input ../data/ancestor-R1.fastq.trimmed.gz > ancestor-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 command.

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

```
C       M02810:197:000000000-AV55U:1:1101:10078:18384/1 2157    151     0:99 2157:1 0:1 215
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 1
C       M02810:197:000000000-AV55U:1:1101:11705:24355/1 2157    151     0:1 2157:1 0:119
```

Each sequence classified by Kraken 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 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 manual can be accessed here.

1.4.6 Investigate taxa

We can use the webpage NCBI TaxIdentifier to quickly get the names to the taxonomy identifier. However, this is impractical as we are dealing potentially with many sequences. Kraken 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 ancestor-R1.kraken > ancestor-R1.kraken.report

The first few lines of and 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 ID
6. 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 ancestor-R1.kraken | **sort -n -k5** > ancestor-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 id.

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 ancestor-R1.kraken > ancestor-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__Betabaculovi
M02810:197:000000000-AV55U:1:1101:10852:5722/1  d__Viruses|f__Phycodnaviridae|g__Phaeovirus
M02810:197:000000000-AV55U:1:1101:11429:10330/1 d__Viruses|f__Microviridae|g__Microvirus|s_
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__Alloherpesvi
M02810:197:000000000-AV55U:1:1101:12336:11626/1 d__Viruses|f__Baculoviridae|g__Betabaculovi
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__Betabaculovi
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**.

Visualisation

We use the Krona tools to create a nice interactive visualisation of the taxa content of our sample *[ONDOV2011]*. Fig. 1.17 shows an example (albeit an artificial one) snapshot of the visualisation Krona provides. Fig. 1.17 is a snapshot of the interactive web-page similar to the one we try to create.

Install Krona with:

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

First some house-keeping to make the Krona 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
```

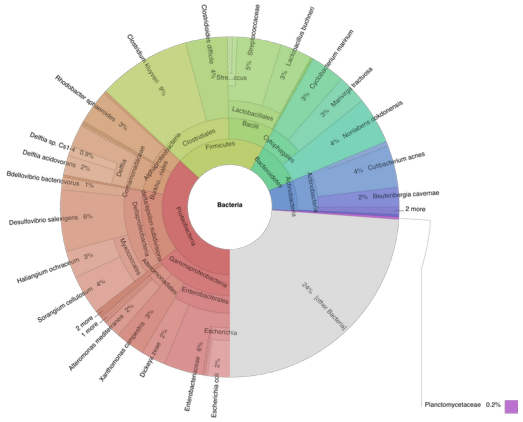

Fig. 1.17: Example of an Krona output webpage.

```
# 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

# now we copy some scripts around, this is neccaesayry as krona installation fails to␣
↪do this
cp -r ~/miniconda3/envs/ngs/opt/krona/scripts ~/miniconda3/envs/py3-kraken/bin/
```

We need to build a taxonomy database for Krona. 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 via a browser.

Now we use the tool `ktImportTaxonomy` from the Krona tools to crate the html web-page:

cat ancestor-R1.kraken | cut -f 2,3 > ancestor-R1.kraken.krona ktImportTaxonomy ancestor-R1.kraken.krona firefox taxonomy.krona.html

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

1.5 NGS - Genome assembly

1.5.1 Preface

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

There is an accompanying lecture for this tutorial:

- Genome Assembly: An Introduction available at figshare (https://dx.doi.org/10.6084/m9.figshare.2972323.v1).

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

1.5.2 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.

1.5.3 Before we start

Lets see how our directory structure looks so far:

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

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

1.5.4 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 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.

Installing the software

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

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

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

Hint: Should you not get it right, try these commands here.

1.5.5 Assembly quality assessment

Assembly statistics

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

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

It is easy with Quast to compare these measures among several assemblies. The program can be used on their website (http://quast.bioinf.spbau.ru/).

We can install it locally with:

```
source activate ngs
conda install quast
```

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

Hint: Should you be unable to run SPAdes on the data, you can download the assemblies here. 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 with regards to the two different assemblies.
2. Which one do you prefer and why?

1.5.6 Assemblathon

Todo

Now that you know the basics for assembling a genome and judging their quality, play with the SPAdes parameters to create the best assembly possible.

Todo

Once you have your final assembly, rename your assembly directory int `spades-final`, e.g. `mv assembly/spades-default assembly/spades-final`.

1.5.7 Further reading

Background on Genome Assemblies

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

Evaluation of Genome Assembly Software

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

1.5.8 Web links

- Lectures for this topic: Genome Assembly: An Introduction
- SPAdes

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

1.6 NGS - Genome annotation

1.6.1 Genome annotation

Genome annotation can be facilitated for example with these programs:

- Genome annotation pipeline RAST (http://rast.nmpdr.org/) [Aziz2008]
- Phage-specific annotation pipeline PHAST (http://phast.wishartlab.com/) [Zhou2011]

Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, Edwards RA, Formsma K, Gerdes S, Glass EM, Kubal M, Meyer F, Olsen GJ, Olson R, Osterman AL, Overbeek RA, McNeil LK, Paarmann D, Paczian T, Parrello B, Pusch GD, Reich C, Stevens R, Vassieva O, Vonstein V, Wilke A, Zagnitko O. **The RAST Server: rapid annotations using subsystems technology.** *'BMC Genomics 2008 Feb 8;9:75 <http://www.biomedcentral.com/1471-2164/9/75>'__*

Zhou Y, Liang Y, Lynch KH, Dennis JJ, Wishart DS. **PHAST: a fast phage search tool.** *'Nucleic Acids Res. 2011 Jul;39(Web Server issue):W347-52. <http://nar.oxfordjournals.org/cgi/pmidlookup?view=long&pmid=21672955>'__*

1.6.2 Web links

- PHAST
- RAST

1.7 NGS - Read mapping

1.7.1 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.

There is an accompanying lecture for this tutorial:

- Genome Assembly: An Introduction available at figshare (https://dx.doi.org/10.6084/m9.figshare.2972323.v1).

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

1.7.2 Learning outcomes

After studying this 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.

1.7.3 Before we start

Lets see how our directory structure looks so far:

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

```
annotation/
assembly/
data/
kraken/
SolexaQA/
SolexaQA++
trimmed/
trimmed-fastqc/
trimmed-solexaqa/
```

1.7.4 Mapping sequence reads to a reference genome

We want to map the sequencing reads to the ancestral reference genome we created in the section *NGS - Genome assembly*. We are going to use the quality trimmed forward and backward DNA sequences of the evolved line and use a program called BWA 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.

Installing the software

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

It is simple to install and use.

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

1.7.5 Bowtie2

Overview

Bowtie2 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 Bowtie2 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 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 default is set to 500 which is often considered too short.

Creating a reference index for mapping

Todo

Create an Bowtie2 index for our reference genome assembly. Attention! Remember which file you need to submit to Bowtie2.

Hint: Should you not get it right, try these commands here.

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 these commands here.

1.7.6 BWA

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

Overview

BWA 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 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 tools we are going to use are shown below:

```
# 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
```

Creating a reference index for mapping

Todo

Create an BWA index for our reference genome assembly. Attention! Remember which file you need to submit to BWA.

Hint: Should you not get it right, try these commands here.

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 these commands here.

1.7.7 The sam mapping file-format

BWA will produce a mapping file in sam-format. Have a look into the sam-file that was created by BWA. A quick overview of the sam-format can be found here and even more information can be found here. 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:

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    ␣
↪TATGGTATCACTTATGGTATCACTTATGGCTATCACTAATGGCTATCACTTATGGTATCACTTATGACTATCAGACGTTATTACTATCAGACGATAAC
↪FHGHHHHHGGGHHHHHHHHHHHHHHHHHHGHHHHHHHHHHHGHHHHGHHHHHHHHGDHHHHHHHHGHHHHGHHHGHHHHHHFHHHHGHHHHIHHHHH
↪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.

1.7.8 Mapping post-processing

Fix mates and compress

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

samtools fixmate -O bam evolved-6.sam 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 information later below to extract specific alignments.

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

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

rm mappings/evolved-6.sam

Sorting

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

convert to bam file and sort samtools sort -O bam -o evolved-6.sorted.bam evolved-6.fixmate.bam

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

1.7.9 Mapping statistics

Stats with SAMtools

Lets get an mapping overview:

samtools flagstat mappings/evolved-6.sorted.bam

Todo

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

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

samtools depth mappings/evolved-6.sorted.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 to make a coverage plot for contig NODE20. Open a R 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. 1.18

Todo

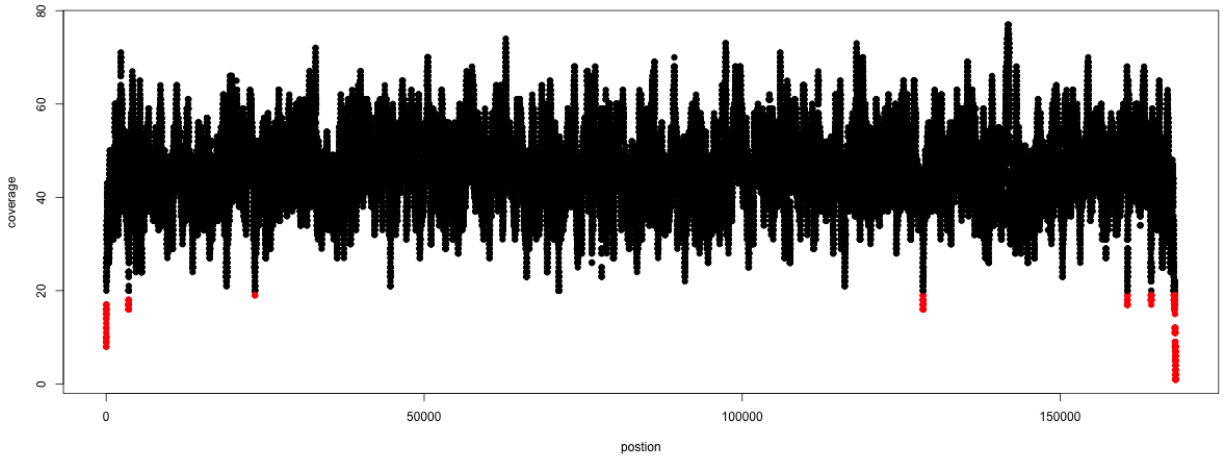

Fig. 1.18: A example coverage plot for a contig with highlighted in red regions with a coverage below 20 reads.

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.

Stats with QualiMap

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

QualiMap 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
```

1.7.10 Sub-selecting reads

It is important to remember that the mapping commands we used above, without additional parameter to sub-select specific alignments (e.g. for Bowtie2 there are options like `--no-mixed`, which suppresses unpaired alignments for paired reads or `--no-discordant`, which suppresses discordant alignments for paired reads, etc.), is 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.

Todo

Explain what concordant and discordant read pairs are? Look at the Bowtie2 manual.

Concordant reads

Here, we select the reads **we will be using for subsequent analyses**. Frist off, we select reads with a mapping quality of at least 20. Furthermore, we select read-pair that have been mapped in a correct manner (same chromosome/contig,

correct orientation to each other).

samtools view -h -b -q 20 -f 2 mappings/evolved-6.sorted.bam > mappings/evolved-6.sorted.concordant.q20.bam

- `-h`: Include the sam header
- `-b`: Output will be bam-format
- `-q 20`: Only extract reads with mapping quality >= 20
- `-f 2`: Only extract correctly paired reads. `-f` extracts alignments with the specified SAM flag set.

Attention: The resulting file of this step will be used in the next section for calling variants.

Unmapped reads

We could decide to use Kraken like in section *NGS - Taxonomic investigation* 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.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 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 evolved-6.sorted.unmapped.bam -fq mappings/evolved-6.sorted.unmapped.R1.fastq -fq2 mappings/evolved-6.sorted.unmapped.R2.fastq

1.8 NGS - Variant calling

1.8.1 Preface

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

There is an accompanying lecture for this tutorial:

- SNPs - GWAS - eQTLs introduction

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

1.8.2 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/
```

1.8.3 Learning outcomes

After studying this tutorial you should be able to:

1. Use tools to call variants based on a reference genome.
2. Identify variants of interests.
3. Understand how the variants might affect the observed biology.

1.8.4 Installing necessary software

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

```
# activate the env
source activate ngs

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

1.8.5 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 ina directory `assembly/spades_final`, use SAMtools to do this:

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

Furthermore we need to pre-process our mapping files a bit further and creaee a bam-index file (`.bai`) for each o the bam-files, e.g.:

bamtools index -in mappings/evolved-6.sorted.bam

Lets also create a new directory for the variants:

```
mkdir variants
```

1.8.6 Calling variants

SAMtools mpileup

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

We first pile up all the reads samtools mpileup -g -f assembly/spades_final/scaffolds.fasta mappings/evolved-6.sorted.bam > variants/evolved-6.mpileup.bcf # Now we call the variants bcftools view -c -v variants/evolved-6.mpileup.bcf > variants/evolved-6.mpileup.vcf

freebayes

Now we can do some variant calling with another tool called freebayes. Given a reference genome scaffold file in fasta-format, e.g. `scaffolds.fasta` and the index in `.fai` format and a mapping file (e.g. "evolved-6.sorted.bam") and a mapping index, we can call freebayes 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.bam > variants/evolved-6.freebayes.vcf

This will result in a variants file "evolved-6.freebayes.vcf".

1.8.7 Post-processing

Understanding the output files (.vcf)

Lets look at a vcf-file:

first 10 lines, which are part of the header cat variants/evolved-6.freebayes.vcf | head

```
##fileformat=VCFv4.1
##fileDate=20161122
##source=freeBayes v1.0.2-29-g41c1313
##reference=genome/scaffolds.fasta
##contig=<ID=NODE_1_length_1394677_cov_15.3771,length=1394677>
##contig=<ID=NODE_2_length_1051867_cov_15.4779,length=1051867>
##contig=<ID=NODE_3_length_950567_cov_15.4139,length=950567>
##contig=<ID=NODE_4_length_925223_cov_15.3905,length=925223>
##contig=<ID=NODE_5_length_916389_cov_15.4457,length=916389>
##contig=<ID=NODE_6_length_772252_cov_15.4454,length=772252>
```

Lets look at the variants:

remove header lines and look at top 4 entires cat variants/evolved-6.freebayes.vcf | egrep -v '##' | head -4

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  unknown
NODE_1_length_1394677_cov_15.3771       137621  .       T       C       76.5197 .
↪  AB=0.318182;ABP=9.32731;AC=1;AF=0.5;AN=2;AO=7;CIGAR=1X;DP=22;DPB=22;DPRA=0;EPP=18.
↪2106;EPPR=4.31318;GTI=0;LEN=1;MEANALT=1;MQM=56.1429;MQMR=56.4;NS=1;NUMALT=1;ODDS=17.
↪6193;PAIRED=1;PAIREDR=1;PAO=0;PQA=0;PQR=0;PRO=0;QA=268;QR=540;RO=15;RPL=0;RPP=18.
↪2106;RPPR=6.62942;RPR=7;RUN=1;SAF=7;SAP=18.2106;SAR=0;SRF=12;SRP=14.7363;SRR=3;
↪TYPE=snp       GT:DP:DPR:RO:QR:AO:QA:GL    0/1:22:22,7:15:540:7:268:-17.3644,0,-42.
↪2185
NODE_1_length_1394677_cov_15.3771       568696  .       G       A       1269.62 .
↪  AB=0;ABP=0;AC=2;AF=1;AN=2;AO=38;CIGAR=1X;DP=38;DPB=38;DPRA=0;EPP=3.23888;EPPR=0;
↪GTI=0;LEN=1;MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=57.2844;PAIRED=1;PAIREDR=0;
↪PAO=0;PQA=0;PQR=0;PRO=0;QA=1438;QR=0;RO=0;RPL=20;RPP=3.23888;RPPR=0;RPR=18;RUN=1;
↪SAF=20;SAP=3.23888;SAR=18;SRF=0;SRP=0;SRR=0;TYPE=snp      GT:DP:DPR:RO:QR:AO:QA:GL
↪      1/1:38:38,38:0:0:38:1438:-129.701,-11.4391,0
```

```
NODE_1_length_1394677_cov_15.3771       612771  .       T       C       60.7485 .
↪   AB=0.3;ABP=9.95901;AC=1;AF=0.5;AN=2;AO=6;CIGAR=1X;DP=20;DPB=20;DPRA=0;EPP=4.45795;
↪EPPR=8.59409;GTI=0;LEN=1;MEANALT=1;MQM=49.5;MQMR=54.3571;NS=1;NUMALT=1;ODDS=13.9879;
↪PAIRED=1;PAIREDR=1;PAO=0;PQA=0;PQR=0;PRO=0;QA=223;QR=540;RO=14;RPL=6;RPP=16.0391;
↪RPPR=33.4109;RPR=0;RUN=1;SAF=4;SAP=4.45795;SAR=2;SRF=4;SRP=8.59409;SRR=10;TYPE=snp
↪   GT:DP:DPR:RO:QR:AO:QA:GL        0/1:20:20,6:14:540:6:223:-12.5734,0,-40.0605
```

Statistics and filter

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

get statistics rtg vcfstats variants/evolved-6.freebayes.vcf

only keep entries with qual of min 30 rtg vcffilter -q 30 -i variants/evolved-6.freebayes.vcf -o variants/evolved-6.freebayes-q30.vcf

look at stats for filtered rtg vcfstats variants/evolved-6.freebayes-q30.vcf

Finding variants of interest (VAI)

Things to consider when looking for VAI:

- 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 where mapped here correctly?
- The location of the SNP.
  - SNPs in larger contigs probably more interesting than in tiny contigs.
  - Does the SNP overlap a coding region in the genome annotation?
- The type of SNP.

Overlap variants with genes

1.9 Quick command reference

1.9.1 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
```

1.9.2 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

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

# activate env
source activate [name]

# deavtivate env
source deactivate
```

1.10 Downloads

1.10.1 Tools

- Miniconda installer: EXTERNAL | INTERNAL
- Minikraken database: EXTERNAL | INTERNAL
- Krona taxonomy database: INTERNAL
- SolexaQA++: INTERNAL

1.10.2 Data

- Data we want to analyse: INTERNAL

1.10.3 Software

```
conda install fastqc     # quality assessment
conda install sickle

conda install spades     # assembler
conda install quast      # QA assemblies

conda install bwa        # mapper

conda install samtools   # Working with mapping files
conda install bamtools   # Working with mapping files
conda install freebayes  # SNP caller
conda install bcftools   # working with SNP files
conda install igvtools   # for visualisation
conda install krona
conda install kraken-all
```

1.11 References

[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. 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

[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

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

[HUNT2014] Hunt M, Newbold C, Berriman M, Otto TD. A comprehensive evaluation of assembly scaffolding tools. Genome Biol. 2014 Mar 3;15(3):R42. doi: 10.1186/gb-2014-15-3-r42.

[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

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

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

[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

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

[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.