Paleogenomics Documentation

Release 0.1.0

Claudio Ottoni

Nov 10, 2020

Contents

1 Contents **3**
1.1 List of Tools . . . 3
1.2 Quality filtering of reads . . . 4
1.3 Metagenomic screening of shotgun data . . . 6
1.4 Alignment of reads to a reference genome . . . 15
1.5 Variant calling and visualization . . . 24
1.6 Filtering, annotating and combining SNPs . . . 26
1.7 DO-IT-YOURSELF . . . 29
1.8 R session . . . 29

Welcome to the 3rd edition of the “Physalia Paleogenomics” course, online version 16-20 November 2020.

CHAPTER 1

Contents

1.1 List of Tools

Data preprocessing

- AdapterRemoval: https://github.com/MikkelSchubert/adapterremoval

Metagenomics screening of shotgun data

- kraken: https://ccb.jhu.edu/software/kraken/
- kraken2: https://ccb.jhu.edu/software/kraken2/index.shtml
- krona: https://github.com/marbl/Krona/wiki

Reads alignment and variants calling

- bwa: https://github.com/lh3/bwa
- gatk: https://software.broadinstitute.org/gatk/
- samtools: http://www.htslib.org/
- bcftools: http://www.htslib.org/

Filtering and manipulating bam files

- picard: https://broadinstitute.github.io/picard/
- dedup: https://github.com/apeltzer/DeDup

aDNA deamination detection and rescalinghttps://github.com/Amine-Namouchi/snpToolkit

- mapDamage: https://ginolhac.github.io/mapDamage/

Creating summary reports

- fastqc: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
- Qualimap: http://qualimap.bioinfo.cipf.es/
- BAMStats: http://bamstats.sourceforge.net

- MultiQC: http://multiqc.info/

Integrative genomic viewer

- IGV: http://software.broadinstitute.org/software/igv/

SNPs annotation and post-processing

- snpToolkit: https://github.com/Amine-Namouchi/snpToolkit

Phylogeny

- IQ-TREE: http://www.iqtree.org/
- Figtree: http://tree.bio.ed.ac.uk/software/figtree

1.2 Quality filtering of reads

1.2.1 Reads quality control

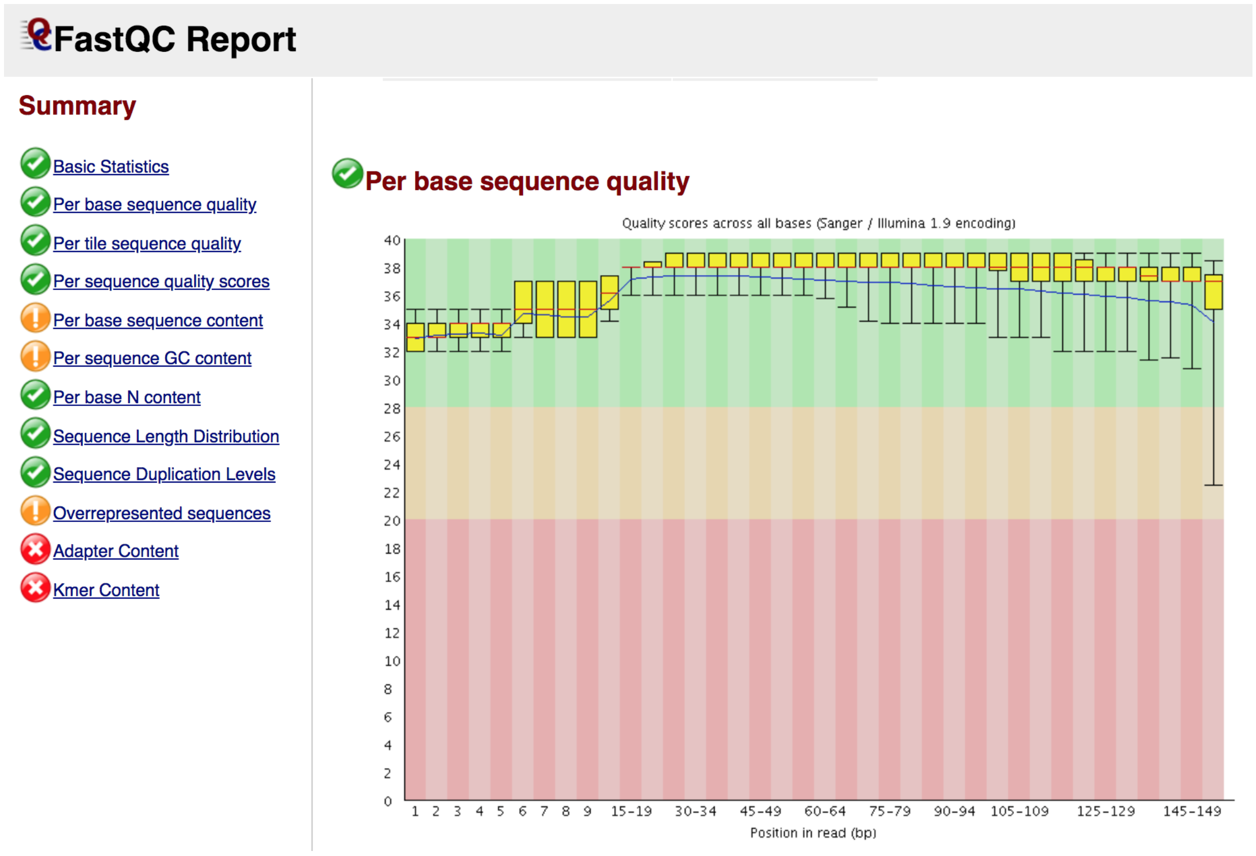
The first step is the quality-control of the reads generated by the sequencing platform in the `fastq` file format. To do that, we will use FastQC, which provides a modular set of analyses that you can use to have a first impression of whether your data has any problems of which you should be aware before doing any further analysis. To run FastQC type the following command:

```
fastqc filename.fastq.gz
```

To analyze multiple `fastq` files you can run `FastQC` as follows:

```
fastqc *.fastq.gz
```

At the end of the analysis, FastQC generates for each input file a summary report, like in the screenshot below:


FastQC Report
Summary
Basic Statistics
Per base sequence quality
Per tile sequence quality
Per sequence quality scores
Per base sequence content
Per sequence GC content
Per base N content
Sequence Length Distribution
Sequence Duplication Levels
Overrepresented sequences
Adapter Content
Kmer Content
Per base sequence quality
Quality scores across all bases (Sanger / Illumina 1.9 encoding)
Position in read (bp)

Note:

- You can download the reports of FastQC (and any other file) in your laptop with the command `scp` (Secure Copy), which allows files to be copied to, from, or between different hosts. It uses ssh for data transfer and provides the same authentication and same level of security as ssh. For example, to copy from a remote host (our server) to your computer:

```
scp username@remotehost:/full_path_to_file /some/local/directory
```

- To copy a folder you need to call the option `-r`

```
scp -r username@remotehost:/full_path_to_file /some/local/directory
```

- If you are using a `pem` file to connect to the server, you have to use in order to download the files:

```
scp -i filename.pem -r username@remotehost:/full_path_to_file /some/local/
↪directory
```

1.2.2 Reads quality filtering

Reads filtering is a crucial step as it will affect all downstream analyses. One of the important things to do is to trim the adapters that were used during the preparation of the genomic libraries. For this step we will use the program AdapterRemoval, which performs adapter trimming of sequencing reads and subsequent merging (collapse) of paired-end reads with negative insert sizes (an overlap between two sequencing reads derived from a single DNA fragment) into a single collapsed read. Here we have single-end reads, so we are going to just trim the adapters:

```
AdapterRemoval --file1 filename.fastq.gz --basename filename --minlength 30 --trimns -
↪-trimqualities --gzip
```

Here some of the options of AdapterRemoval:

Option	Function
-file1 *string*	Forward reads input file(s) in fastq(.gz) file format. Required option (single-end reads).
-file2 *string*	Reverse reads input file(s) in fastq(.gz) file format.
–basename *string*	Default prefix for all output files for which no filename was explicitly set [current: your_output]
–adapter1 *sequence*	Adapter sequence expected to be found in mate 1 reads [current: AGATCGGAAGAGCA-CACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG]
–adapter2 *sequence*	Adapter sequence expected to be found in mate 2 reads [current: AGATCGGAAGAGCGTCGTG-TAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT]
–trimns	If set, trim ambiguous bases (N) at 5'/3' termini [current: off]
–trimqualities	If set, trim bases at 5'/3' termini with quality scores <= to **–minquality** value [current: off]
–minquality *integer*	PHRED inclusive minimum values; see **–trimqualities** for details [current: 2]
–minlength *integer*	Reads shorter than this length are discarded following trimming [current: 15].
–collapse	When set, paired ended read alignments of **–minalignmentlength** or more bases are combined into a single consensus sequence, representing the complete insert
–minalignmentlength *integer*	If **–collapse** is set, paired reads must overlap at least this number of bases to be collapsed, and single-ended reads must overlap at least this number of bases with the adapter to be considered complete template molecules [current: 11].

Note:

- Here we are using a Single-End library, for paired-end libraries the command to clip the adapter and merge the read pairs is:

```
AdapterRemoval --file1 filename_R1.fastq --file2 filename_R2.fastq --basename␣
↪filename --minlength 30 --trimns --trimqualities --collapse --gzip
```

- Several tools can be used for reads pre-processing and filtering, for example: ClipAndMerge, leeHom, Atropos, fastp.

After reads filtering open your adapter-trimmed `fastq` file again in FastQC and see the differences before (the two original paired-end reads files) and after (the collapsed reads file) adapter trimming.

1.3 Metagenomic screening of shotgun data

1.3.1 Kraken

In this hands-on session we will use Kraken to screen the metagenomic content of a DNA extract after shotgun sequencing. A Kraken database is a directory containing at least 4 files:

- **database.kdb**: Contains the k-mer to taxon mappings
- **database.idx**: Contains minimizer offset locations in database.kdb
- **taxonomy/nodes.dmp**: Taxonomy tree structure + ranks
- **taxonomy/names.dmp**: Taxonomy names

Minikraken

We will first use a pre-built 8 GB Kraken database, called Minikraken, constructed from complete dusted bacterial, archaeal, and viral genomes in RefSeq (as of October 2017). You can download the pre-built Minikraken database from the website with `wget`, and extract the archive content with `tar`:

```
wget https://ccb.jhu.edu/software/kraken/dl/minikraken_20171101_8GB_dustmasked.tgz
tar -xvzf minikraken_20171101_8GB_dustmasked.tgz
```

Then, we can run the taxonomic assignation of the reads in our sample with the `kraken` command

```
kraken --db minikraken_20171101_8GB_dustmasked --fastq-input filename.gz --gzip-
↪compressed --output filename.kraken
```

Some of the options available in Kraken:

Option	Function
–db <string>	Path to the folder (database name) containing the database files.
–output <string>	Print output to filename.
–threads <integer>	Number of threads (only when multiple cores are used).
–fasta-input	Input is FASTA format.
–fastq-input	Input is FASTQ format.
–gzip-compressed	Input is gzip compressed.

Create report files

Once the taxonomic assignation is done, from the Kraken output file we create a report of the analysis by running the `kraken-report` script. Note that the database used must be the same as the one used to generate the output file in the command above. The output file is a tab-delimited file with the following fields, from left to right:

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

Notice that we will have to redirect the output to a file with >.

```
kraken-report --db Minikraken filename.kraken > filename.kraken.report
```

Note: We can use a `for` loop to make the taxonomic assignation and create the report file for multiple samples. Notice the assignation of variables `filename` and `fname` to return output files named after the sample.

```
for i in *.fastq
do
  filename=$(basename "$i")
  fname="${filename%.fastq}"
  kraken --db Minikraken --threads 4 --fastq-input $i --output /${fname}.kraken
  kraken-report --db Minikraken ${fname}.kraken > ${fname}.kraken.report
done
```

Visualization of data with Krona

Finally, we can visualize the results of the Kraken analysis with Krona, which disaplys hierarchical data (like taxonomic assignation) in multi-layerd pie charts. The interactive charts created by Krona are in the `html` format and can be viewed with any web browser. We will convert the kraken output in html format using the program `ktImportTaxonomy`, which parses the information relative to the query ID and the taxonomy ID.

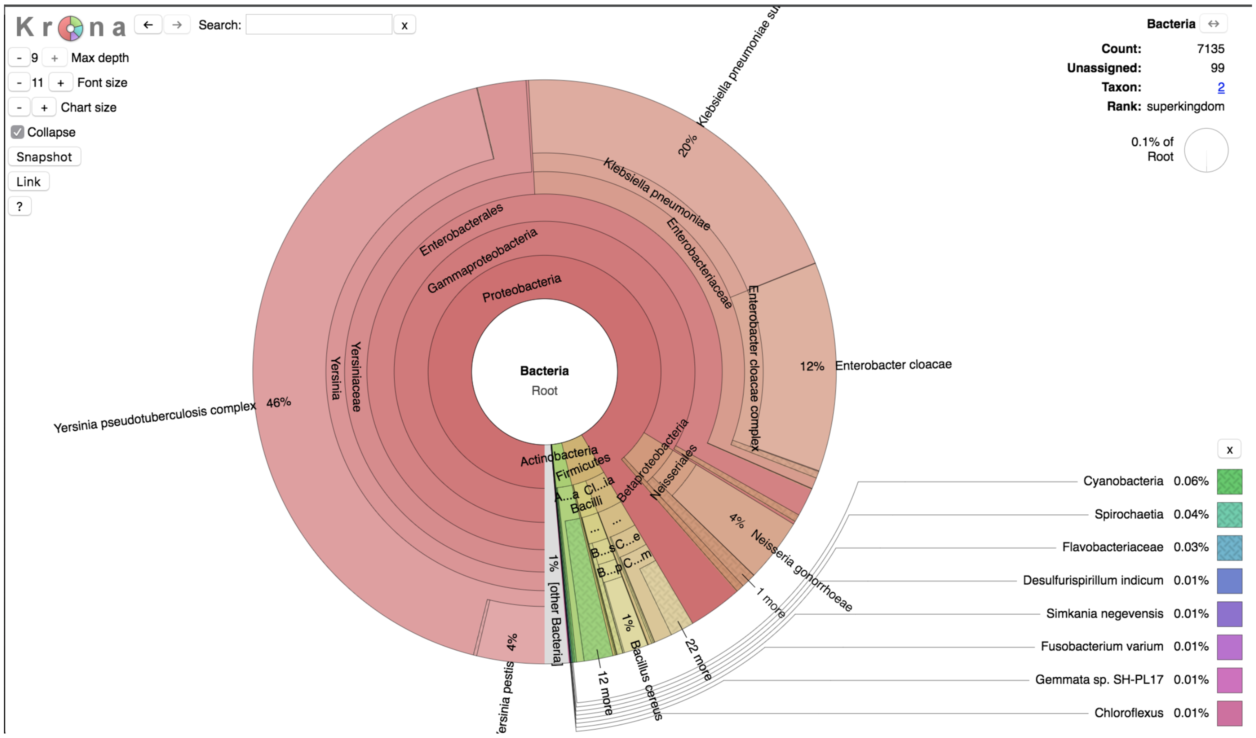
```
ktImportTaxonomy -q 2 -t 3 filename.kraken -o filename.kraken.html
```

Some of the options available in ktImportTaxonomy:

Option	Function
-q <integer>	Column of input files to use as query ID.
-t <integer>	Column of input files to use as taxonomy ID.
-o <string>	Output file name.

Note: If you want to analyze multiple kraken files from various samples you view the results in one single html file running `ktImportTaxonomy` as follows:

```
ktImportTaxonomy -q 2 -t 3 filename_1.kraken filename_2.kraken ... filename_n.kraken -
↪o all_samples.kraken.html
```



Bacteria

Count: 7135
Unassigned: 99
Taxon: 2
Rank: superkingdom

Building a Kraken standard database (on HPC clusters)

The pre-built Minikraken database is useful for a quick metagenomic screening of shotgun data. However, by building larger databases (i.e. a larger set of k-mers gathered) we may increase the sensitivity of the analysis. One option is to build the Kraken standard database. To create this database we use the command `kraken-build`, which downlads the `RefSeq` complete genomes for the bacterial, archaeal, and viral domains, and builds the database.

```
kraken-build --standard --db standardkraken.folder
```

Note:

- Usage of the database will require users to keep only the `database.idx`, `database.kdb`, `taxonomy/nodes.dmp` and `taxonomy/names.dmp` files. During the database building process some intermediate file are created that may be removed afterwards with the command:

  ```
  kraken-build --db standardkraken.folder --clean
  ```

- The downloaded RefSeq genomes require 33GB of disk space. The build process will then require approximately 450GB of additional disk space. The final `database.idx`, `database.kdb`, and `taxonomy/` files require 200 Gb of disk space, and running one sample against such database requires 175 Gb of RAM.

Building a Kraken custom database (on HPC clusters)

Building kraken custum databases is computationally intensive. You will find a ready to use database. Kraken also allows creation of customized databases, where we can choose which sequences to include and the final size of the database. For example if you do not have the computational resources to build and run analyses with a full database of bacterial genomes (or you don't need to), you may want to build a custom database with only the genomes needed for your application.

1. First of all we choose a name for our database and we create a folder with that name using `mkdir`. Let's call the database `CustomDB`. This will be the name used in all the dollowing commands after the `--db` option.

2. Download NCBI taxonomy files (the sequence ID to taxon map, the taxonomic names and tree information) with `kraken-build --download-taxonomy`. The taxonomy files are necessary to associate a taxon to the sequence identifier (the GI number in NCBI) of the `fasta` sequences composing our database. For this reason we will build our database only with sequences from the NCBI RefSeq. For more information on the NCBI taxonomy visit click here. This command will create a sub-folder `taxonomy/` inside our CustomDB folder:

   ```
   kraken-build --download-taxonomy --threads 4 --db CustomDB
   ```

3. Install a genomic library. RefSeq genomes in fasta file from five standard groups are made easily available in Kraken with the command **kraken-build –download-library**:

   - *bacteria* : RefSeq complete bacterial genomes
   - *archaea* : RefSeq complete archaeal genomes
   - *plasmid* : RefSeq plasmid sequences
   - *viral* : RefSeq complete viral genomes
   - *human* : GRCh38 human genome

   The following command will download all the RefSeq bacterial genomes (33Gb size) and create a folder `library/` with a sub-folder `bacteria/` inside your CustomDB folder:

   ```
   kraken-build --download-library bacteria --threads 4 --db CustomDB
   ```

4. We can add any sort of RefSeq `fasta` sequences to the library with `kraken-build --add-to-library`. For example we will add to the library of bacterial genomes the RefSeq sequences of mitochodrial genomes. The sequences will be inside the sub-folder `added/`.

   ```
   kraken-build --add-to-library mitochondrion.1.1.genomic.fna --threads 4 --db␣
   ↪CustomDB
   kraken-build --add-to-library mitochondrion.2.1.genomic.fna --threads 4 --db␣
   ↪CustomDB
   ```

   Note: If you have several `fasta` files to add you can use a `for` loop:

   ```
   for i in *.fasta
   do
     kraken-build --add-to-library $i --threads 4 --db CustomDB
   done
   ```

5. When analyzing a metagenomics sample using a `Kraken` database the primary source of false positive hits is represented by low-complexity sequences in the genomes themselves (e.g., a string of 31 or more consecutive A's). For this reason, once gathered all the genomes that we want to use for our custom database, low-complexity regions have to be 'dusted'. The program `dustmasker` from Blast+ identifies low-complexity regions and **soft-mask** them (the corresponding sequence is turned to lower-case letters). With a `for` loop we run dustmasker on each `fasta` file present in the library folder, and we will pipe (`|`) to dustmasker a `sed` command to replace the low-complexity regions (lower-case) with Ns. Notice that the output is redirected (>) to a temporary file, which is afterwards renamed to replace the original file `fasta` file with the command `mv`.

```
for i in `find CustomDB/library \( -name '*.fna' -o -name '*.ffn' \)`
do
  dustmasker -in $i -infmt fasta -outfmt fasta | sed -e '/>/!s/a\|c\|g\|t/N/g' >␣
↪tempfile
  mv -f tempfile $i
done
```

Some of the options available in Dustmasker:

Option	Function
-in <string>	input file name
-infmt <string>	input format (e.g. fasta)
-outfmt8 <string>	output format (fasta)

6. Finally, we build the database with `kraken-build`. With this command, Kraken uses all the masked genomes contained in the library (bacteria and mtDNA RefSeq) to create a database of 31 bp-long k-mers. We can choose the size of our custom database (hence the number of k-mers included, and the sensitivity) with the `--max-db-size` option (8 Gb here).

```
kraken-build --build --max-db-size 8 --db CustomDB
```

Taxonomic assignation with Kraken custom database

Once our custom database is built we can run the command for taxonomic assignation of DNA reads agaisnt the custom database, as in section 1.1 and 1.2.

```
kraken --db CustomDB --fastq-input merged.fastq.gz --gzip-compressed --output sample.
↪kraken
kraken-report --db CustomDB sample.kraken > sample.kraken.report
```

Or, again, we can loop the commands if we have various samples.

```
for i in *.fastq
do
  filename=$(basename "$i")
  fname="${filename%.fastq}"
  kraken --db CustomDB --threads 4 --fastq-input $i --output ${fname}.kraken
  kraken-report --db CustomDB ${fname}.kraken > ${fname}.kraken.report
done
```

Finally, we can visualize the results of the Kraken analysis with `Krona`. Run `ktImportTaxonomy` to generate the `html` file and open it in a web browser to notice the difference with the analysis done with Minikraken.

```
ktImportTaxonomy -q 2 -t 3 sample.kraken -o sample.kraken.html
```

1.3.2 Kraken2

The first version of Kraken use a large indexed and sorted list of k-mer/LCA pairs as its database, which may be problematic for users due to the large memory requirements. For this reason Kraken 2 was developed. Kraken 2 is fast and requires less memory, BUT the database false positive errors occur in less than 1% of queries (confidence scoring thresholds can be used to call out to detect them). The default (or standard) database size is 29 GB (as of Jan. 2018, in Kraken 1 the standard database is about 200 GB!), and you will need slightly more than that in RAM if you want to build it. By default, Kraken 2 will attempt to use the dustmasker or segmasker programs provided as part of NCBI's BLAST suite to mask low-complexity regions.

A Kraken 2 database is a directory containing at least 3 files:

- **hash.k2d**: Contains the minimizer to taxon mappings
- **opts.k2d**: Contains information about the options used to build the database
- **taxo.k2d**: Contains taxonomy information used to build the database

Other files may also be present as part of the database build process, and can, if desired, be removed after a successful build of the database.

Minikraken2

You can download the pre-built Minikraken2 database (containing bacteria, viral and archaea sequences) from the website with `wget`, and extract the archive content with `tar`:

```
wget ftp://ftp.ccb.jhu.edu/pub/data/kraken2_dbs/minikraken2_v1_8GB_201904_UPDATE.tgz
tar -xvzf minikraken2_v1_8GB_201904_UPDATE.tgz
```

To classify the reads in a `fastq` file against the Minikraken2 database, you can run this command:

```
kraken2 --db minikraken2_v1_8GB filename.fastq.gz --gzip-compressed --output filename.
↪kraken --report filename.kraken.report
```

Some of the options available in Kraken 2:

Option	Function
–use-names	Print scientific names instead of just taxids
–gzip-compressed	Input is gzip compressed
–report <string>	Print a report with aggregrate counts/clade to file
–threads	Number of threads (default: 1)

Note:

- In Kraken 2 you can generate the reports file by typing the `--report` option (followed by a name for the report file to generate) in the command used for the classification. In Kraken 1, report files are generated with a specific command, after the classification (section 3.1.2: *Create report files*).
- In order to run later Krona, the Kraken output file must contain taxids, and not scientific names. So if you want to run Krona do not call the option `--use-names`.

The Kraken 2 report format, like Kraken 1, is tab-delimited with one line per taxon. There are six fields, from left to right:

1. Percentage of fragments covered by the clade rooted at this taxon
2. Number of fragments covered by the clade rooted at this taxon
3. Number of fragments assigned directly to this taxon

4. A rank code, indicating (U)nclassified, (R)oot, (D)omain, (K)ingdom, (P)hylum, (C)lass, (O)rder, (F)amily, (G)enus, or (S)pecies. Taxa that are not at any of these 10 ranks have a rank code that is formed by using the rank code of the closest ancestor rank with a number indicating the distance from that rank. E.g., "G2" is a rank code indicating a taxon is between genus and species and the grandparent taxon is at the genus rank.
5. NCBI taxonomic ID number
6. Indented scientific name

To visualize the results of the classification in multi-layerd pie charts, use `Krona`, as described in the section 3.1.3: *Visualization of data with Krona*

Kraken 2 Custom Database

We have already created a custom database to use in this hands-on session so we can go straight to the classification (step 4). However, we report here all the commands to build a Kraken2 database (steps 1-3).

1. The first step is to create a new folder that will contain your custom database (choose an appropriate name for the folder-database, here we will call it `CustomDB`). Then we have to install a taxonomy. Usually, you will use the NCBI taxonomy. The following command will download in the folder `/taxonomy` the accession number to taxon maps, as well as the taxonomic name and tree information from NCBI:

   ```
   kraken2-build --download-taxonomy --db CustomDB
   ```

2. Install one or more reference libraries. Several sets of standard genomes (or proteins) are available, which are constantly updated (see also the Kraken website).

   - archaea: RefSeq complete archaeal genomes/proteins
   - bacteria: RefSeq complete bacterial genomes/proteins
   - plasmid: RefSeq plasmid nucleotide/protein sequences
   - viral: RefSeq complete viral genomes/proteins
   - human: GRCh38 human genome/proteins
   - fungi: RefSeq complete fungal genomes/proteins
   - plant: RefSeq complete plant genomes/proteins
   - protozoa: RefSeq complete protozoan genomes/proteins
   - nr: NCBI non-redundant protein database
   - nt: NCBI non-redundant nucleotide database
   - env_nr: NCBI non-redundant protein database with sequences from large environmental sequencing projects
   - env_nt: NCBI non-redundant nucleotide database with sequences from large environmental sequencing projects
   - UniVec: NCBI-supplied database of vector, adapter, linker, and primer sequences that may be contaminating sequencing projects and/or assemblies
   - UniVec_Core: A subset of UniVec chosen to minimize false positive hits to the vector database

   You can select as many libraries as you want and run the following command, which will download the reference sequences in the folder `/library`, as follows:

```
kraken2-build --download-library bacteria --db CustomDB
kraken2-build --download-library viral --db CustomDB
kraken2-build --download-library plasmid --db CustomDB
kraken2-build --download-library fungi --db CustomDB
```

In a custom database, you can add as many `fasta` sequences as you like. For exmaple, you can download the organelle genomes in `fasta` files from the RefSeq website with the commands `wget`:

```
wget ftp://ftp.ncbi.nlm.nih.gov/refseq/release/mitochondrion/mitochondrion.1.1.
↪genomic.fna.gz
wget ftp://ftp.ncbi.nlm.nih.gov/refseq/release/mitochondrion/mitochondrion.2.1.
↪genomic.fna.gz
```

The downloaded files are in compressed in the `gz` format. To unzip them run the `gunzip` command:

```
gunzip mitochondrion.1.1.genomic.fna.gz
gunzip mitochondrion.2.1.genomic.fna.gz
```

Then you can add the `fasta` files to your library, as follows:

```
kraken2-build --add-to-library mitochondrion.1.1.genomic.fna --db CustomDB
kraken2-build --add-to-library mitochondrion.2.1.genomic.fna --db CustomDB
```

3. Once your library is finalized, you need to build the database (here we set a maximum database size of 8 GB (you must indicate it in bytes!) with the `--max-db-size` option).

```
kraken2-build --build --max-db-size 8000000000 --db CustomDB
```

Warning: Kraken 2 uses two programs to perform low-complexity sequence masking, both available from NCBI: `dustmasker`, for nucleotide sequences, and `segmasker`, for amino acid sequences. These programs are available as part of the NCBI BLAST+ suite. If these programs are not installed on the local system and in the user's PATH when trying to use `kraken2-build`, the database build will fail. Users who do not wish to install these programs can use the `--no-masking` option to kraken2-build in conjunction with any of the `--download-library`, `--add-to-library`, or `--standard` options; use of the `--no-masking` option will skip masking of low-complexity sequences during the build of the Kraken 2 database.

The `kraken2-inspect` script allows users to gain information about the content of a Kraken 2 database. You can pipe the command to `head`, or `less`.

```
kraken2-inspect --db path/to/dbfolder | head -5
```

4. Finally, we can run the classification of the reads against the custom database with the `kraken2` command:

```
kraken2 --db CustomDB filename.fastq.gz --gzip-compressed --output filename.
↪kraken --report filename.report
```

5. To visualize the results of the classification in multi-layerd pie charts, use `Krona`, as described in the section 3.1.3: *Visualization of data with Krona*

Note: Recently, a novel metagenomics classifier, KrakenUniq, has been developed to reduce false-positive identifications in metagenomic classification. KrakenUniq combines the fast k-mer-based classification of Kraken with an efficient algorithm for assessing the coverage of unique k-mers found in each species in a dataset. On various test datasets, KrakenUniq gives better recall and precision than other methods and effectively classifies and distinguishes pathogens with low abundance from false positives in infectious disease samples.

1.3.3 Mataphlan 3

MetaPhlAn is a computational tool that relies on ~1.1M unique clade-specific marker genes identified from ~100,000 reference genomes (~99,500 bacterial and archaeal and ~500 eukaryotic) to conduct taxonomic profiling of microbial communities (Bacteria, Archaea and Eukaryotes) from metagenomic shotgun sequencing data. MetaPhlAn allows:

- unambiguous taxonomic assignments;
- an accurate estimation of organismal relative abundance;
- species-level resolution for bacteria, archaea, eukaryotes, and viruses;
- strain identification and tracking
- orders of magnitude speedups compared to existing methods.
- metagenomic strain-level population genomics

To basic usage of MetaPhlAn for taxonomic profiling is:

```
metaphlan filename.fastq(.gz) --input_type fastq -o outfile.txt
```

Note: MetaPhlAn relies on BowTie2 to map reads against marker genes. To save the intermediate BowTie2 output use `--bowtie2out`, and for multiple CPUs (if available) use `--nproc`:

```
metaphlan filename.fastq(.gz) --bowtie2out filename.bowtie2.bz2 --nproc 5 --input_
↪type fastq -o output.txt
```

The intermediate BowTie2 output files can be used to run MetaPhlAn quickly by specifying the input (`--input_type bowtie2out`):

```
metaphlan filename.bowtie2.bz2 --nproc 5 --input_type bowtie2out -o output.txt
```

For more information and advanced usage of MetaPhlAn see the manual and the wiki page (available for MetaPhlAn 2 at the moment).

1.4 Alignment of reads to a reference genome

The metagenomic screenig of the shotgun library detected reads assigned to *Yersinia pestis*. The following step is to ascertain that these molecules are authentic. You can do that by mapping your pre-processed `fastq` files (merged and trimmed) to the *Yersinia pestis* CO92 strain reference sequence, available in the RefSeq NCBI database. Here, in order to obtain an optimal coverage for the subsequent variant call, we will run the alignment on a different `fastq` file that we prepared to simulate an enriched library.

1.4.1 Preparation of the reference sequence

Index the reference sequence with bwa

To align the reads to the reference sequence we will use the program BWA, in particular the `BWA aln` algorithm. BWA first needs to construct the **FM-index** for the reference genome, with the command `BWA index`. FM-indexing in Burrows-Wheeler transform is used to efficiently find the number of occurrences of a pattern within a compressed text, as well as locate the position of each occurrence. It is an essential step for querying the DNA reads to the reference sequence. This command generates five files with different extensions: `amb`, `ann`, `bwt`, `pac`, `sa`.

```
bwa index -a is reference.fasta
```

Note: The option `-a` indicates the algorithm to use for constructing the index. For genomes smaller than < 2 Gb use the `is` algorithm. For larger genomes (>2 Gb), use the `bwtsw` algorithm.

Create a reference dictionary

A dictionary file (`dict`) is necessary to run later in the pipeline `GATK RealignerTargetCreator`. A sequence dictionary contains the sequence name, sequence length, genome assembly identifier, and other information about the sequences. To create the `dict` file we use Picard.

```
picard CreateSequenceDictionary R= referece.fasta O= reference.dict
```

Note: In our server environment we can call Picard just by typing the program name. In other environments (including your laptop) you may have to call Picard by providing the full path to the java file `jar` of the program:

```
java -jar /path/to/picard.jar CreateSequenceDictionary R= referece.fasta O= ref.dict
```

Index the reference sequence with Samtools

The reference sequence has to be indexed in order to run later in the pipeline `GATK IndelRealigner`. To do that, we will use Samtools, in particular the tool `samtools faidx`, which enables efficient access to arbitrary regions within the reference sequence. The index file typically has the same filename as the corresponding reference sequece, with the extension `fai` appended.

```
samtools faidx reference.fasta
```

1.4.2 Alignment of the reads to the reference sequence

Alignment of pre-processed reads to the reference genome with BWA aln

To align the reads to the reference genome we will use `BWA aln`, which supports an end-to-end alignment of reads to the reference sequence. The alternative algorithm, `BWA mem` supports also local (portion of the reads) and chimeric alignments (resulting in a larger number of mapped reads than `BWA aln`). `BWA aln` is more suitable for aliging short reads, like expected for ancient DNA samples. The following comand will generate a `sai` file.

```
bwa aln reference.fasta filename.fastq.gz -n 0.1 -l 1000 > filename.sai
```

Some of the options available in `BWA aln`:

Option	Function
-n *number*	Maximum edit distance if the value is an *integer*. If the value is *float* the edit distance is automatically chosen for different read lengths (default=0.04)
-l *integer*	Seed length. If the value is larger than the query sequence, seeding will be disabled.
-o *integer*	Maximum number of gap opens. For aDNA, tolerating more gaps helps mapping more reads (default=1).

Note:

- Due to the particular damaged nature of ancient DNA molecules, carrying deaminations at the molecules ends, we deactivate the `seed-length` option by giving it a high value (e.g. `-l 1000`).
- Here we are aligning reads to a bacterial reference genome. To reduce the impact of spurious alignemnts due to presence bacterial species closely related to the one that we are investigating, we will adopt stringent conditions by decreasing the `maximum edit distance` option (`-n 0.1`). For alignment of DNA reads to the human reference sequence, less stringent conditions can be used (`-n 0.01` and `-o 2`).

Once obtained the `sai` file, we align the reads (`fastq` file) to the reference (`fasta` file) using `BWA samse`, to generate the alignment file `sam`.

```
bwa samse reference.fasta filename.sai filename.fastq.gz -f filename.sam
```

Converting sam file to bam file

For the downstream analyses we will work with the binary (more compact) version of the `sam` file, called `bam`. To convert the `sam` file in `bam` we will use `Samtools view`.

```
samtools view -Sb filename.sam > filename.bam
```

Note: The conversion from `sam` to `bam` can be piped (`|`) in one command right after the alignment step:

```
bwa samse reference.fasta filename.sai filename.fastq.gz | samtools view -Sb - >␣
↪filename.bam
```

To view the content of a `sam` file we can just use standard commands like `head`, `tail`, `less`, while to view the content of a `bam` file (binary format of `sam`) we have to use `Samtools view`:

```
samtools view filename.bam
```

You may want to display on the screen one read/line (scrolling with the spacebar):

```
samtools view filename.bam | less -S
```

while to display just the header of the bam file:

```
samtools view -H filename.bam
```

Sorting and indexing the bam file

To go on with the analysis, we have to sort the reads aligned in the bam file by leftmost coordinates (or by read name when the option -n is used) with Samtools sort. The option -o is used to provide an output file name:

```
samtools sort filename.bam -o filename.sort.bam
```

The sorted bam files are then indexed with Samtools index. Indexes allow other programs to retrieve specific parts of the bam file without reading through each sequence. The following command generates a bai file, a companion file of the bam which contains the indexes:

```
samtools index filename.sort.bam
```

Adding Read Group tags and indexing bam files

A number of predefined tags may be appropriately assigned to specific set of reads in order to distinguish samples, libraries and other technical features. To do that we will use Picard. You may want to use RGLB (library ID) and RGSM (sample ID) tags at your own convenience based on the experimental design. Remember to call Picard from the path of the jar file.

```
picard AddOrReplaceReadGroups INPUT= filename.sort.bam OUTPUT= filename.RG.bam␣
↪RGID=rg_id RGLB=lib_id RGPL=platform RGPU=plat_unit RGSM=sam_id VALIDATION_
↪STRINGENCY=LENIENT
```

Note:

- In some instances, Picard may stop running and return error messages due to conflicts with sam specifications produced by BWA (e.g. "MAPQ should be 0 for unmapped reads"). To suppress this error and allow Picard to continue, we pass the VALIDATION_STRINGENCY=LENIENT options (default is STRICT).
- Read Groups may be also added during the alignment with BWA using the option -R.

Once added the Read Group tags, we index again the bam file:

```
samtools index filename.RG.bam
```

Marking and removing duplicates

Amplification through PCR of genomic libraries leads to duplication formation, hence reads originating from a single fragment of DNA. The MarkDuplicates tool of Picard marks the reads as duplicates when the 5'-end positions of both reads and read-pairs match. A metric file with various statistics is created, and reads are removed from the bam file by using the REMOVE_DUPLICATES=True option (the default option is False, which simply 'marks' duplicate reads keep them in the bam file).

```
picard MarkDuplicates I= filename.RG.bam O= filename.DR.bam M=output_metrics.txt␣
↪REMOVE_DUPLICATES=True VALIDATION_STRINGENCY=LENIENT &> logFile.log
```

Once removed the duplicates, we index again the bam file:

```
samtools index filename.DR.bam
```

Local realignment of reads

The presence of insertions or deletions (indels) in the genome may be responsible of misalignments and bases mismatches that are easily mistaken as SNPs. For this reason, we locally realign reads to minimize the number of mispatches around the indels. The realignment process is done in two steps using two different tools of GATK called with the -T option. We first detect the intervals which need to be realigned with the `GATK RealignerTargetCreator`, and save the list of these intevals in a file that we name `target.intervals`:

```
gatk -T RealignerTargetCreator -R reference.fasta -I filename.DR.bam -o target.
↪intervals
```

Note: Like Picard, in some server environment you can call GATK just by typing the program name. In other environments (also in this server) you have to call GATK by providing the full path to the java `jar` file. Here, the absolute path to the file is `~/Share/Paleogenomics/programs/GenomeAnalysisTK.jar`:

```
java -jar ~/Share/Paleogenomics/programs/GenomeAnalysisTK.jar -T␣
↪RealignerTargetCreator -h
```

Warning: In *version 4* of GATK the indel realigment tools have been retired from the best practices (they are unnecessary if you are using an assembly based caller like **Mutect2** or **HaplotypeCaller**). To use the indel realignment tools make sure to install *version 3* of GATK.

Then, we realign the reads over the intervals listed in the `target.intervals` file with the option `-targetIntervals` of the tool `IndelRealigner` in GATK:

```
java -jar ~/Share/Paleogenomics/programs/GenomeAnalysisTK.jar  -T IndelRealigner -R␣
↪reference.fasta -I filename.RG.DR.bam -targetIntervals target.intervals -o filename.
↪final.bam --filter_bases_not_stored
```

Note:

- If you want, you can redirect the standard output of the command into a `log` file by typing at the end of the command `&> logFile.log`
- The option `--filter_bases_not_stored` is used to filter out reads with no stored bases (i.e. with * where the sequence should be), instead of failing with an error

The final `bam` file has to be sorted and indexed as previously done:

```
samtools sort filename.final.bam -o filename.final.sort.bam
samtools index filename.final.sort.bam
```

Generate flagstat file

We can generate a file with useful information about our alignment with `Samtools flagstat`. This file is a final summary report of the bitwise `FLAG` fields assigned to the reads in the `sam` file.

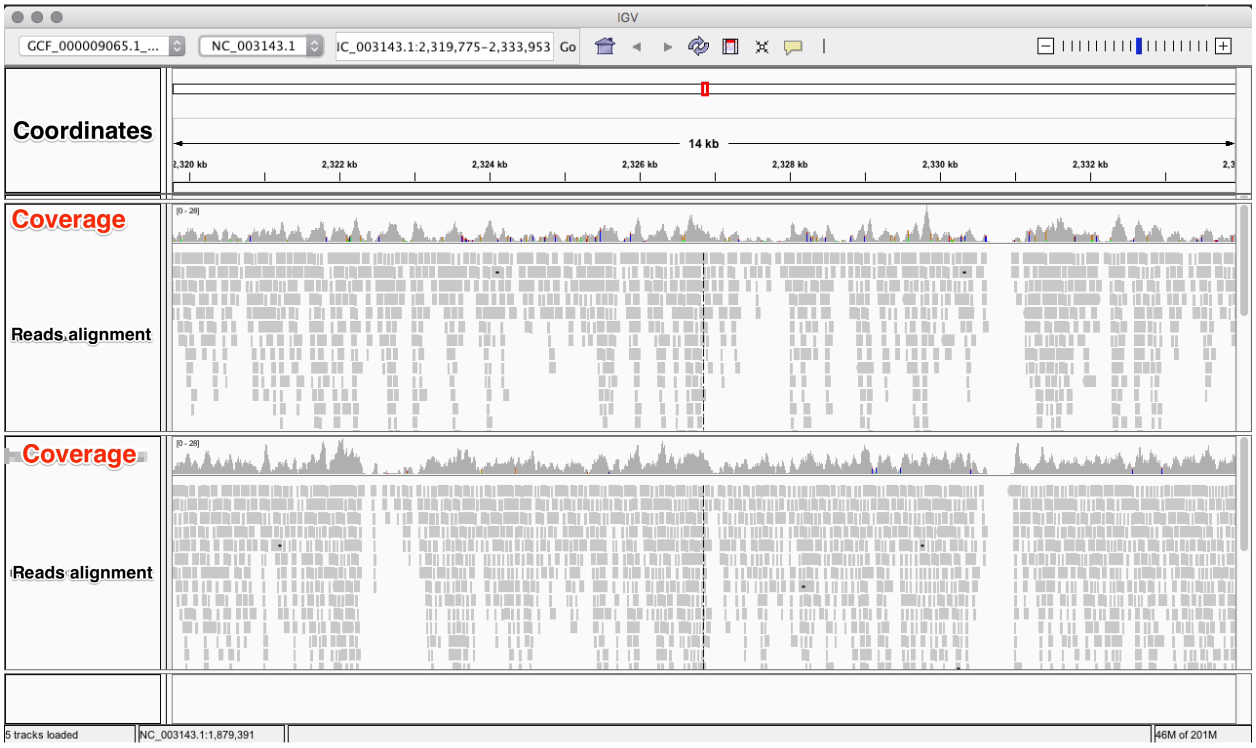
```
samtools flagstat filename.final.sort.bam > flagstat_filename.txt
```

Note:

- You could generate a flagstat file for the two `bam` files before and after refinement and see the differences.
- You can decode each `FLAG` field assigned to a read on the Broad Institute website.

Visualization of reads alignment

Once generated the final `bam` file, you can compare the `bam` files before and after the refinement and polishing process (duplicates removal, realignment around indels and sorting). To do so, we will use the program `IGV`, in which we will first load the reference fasta file from *Genomes –> Load genome from file* and then we will add one (or more) bam files with *File –> Load from file*:

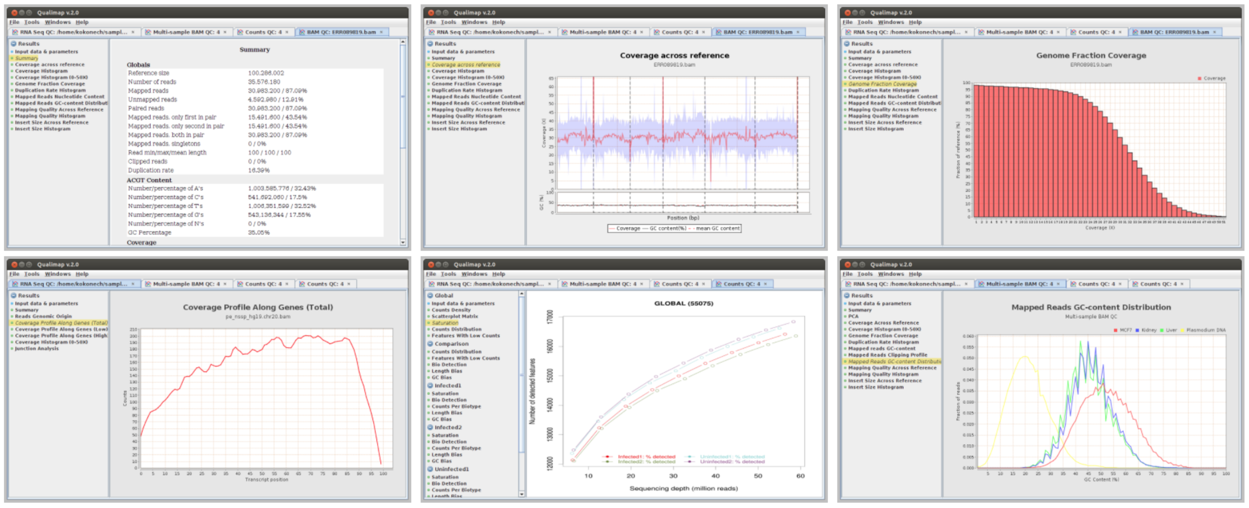


1.4.3 Create mapping summary reports

We will use `Qualimap` to create summary reports from the generated `bam` files. As mentioned in the website, `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 detect biases in the sequencing and/or mapping of the data and eases decision-making for further analysis.

```
qualimap bamqc -c -bam input.bam
```

Here are some screenshots of the outputs:

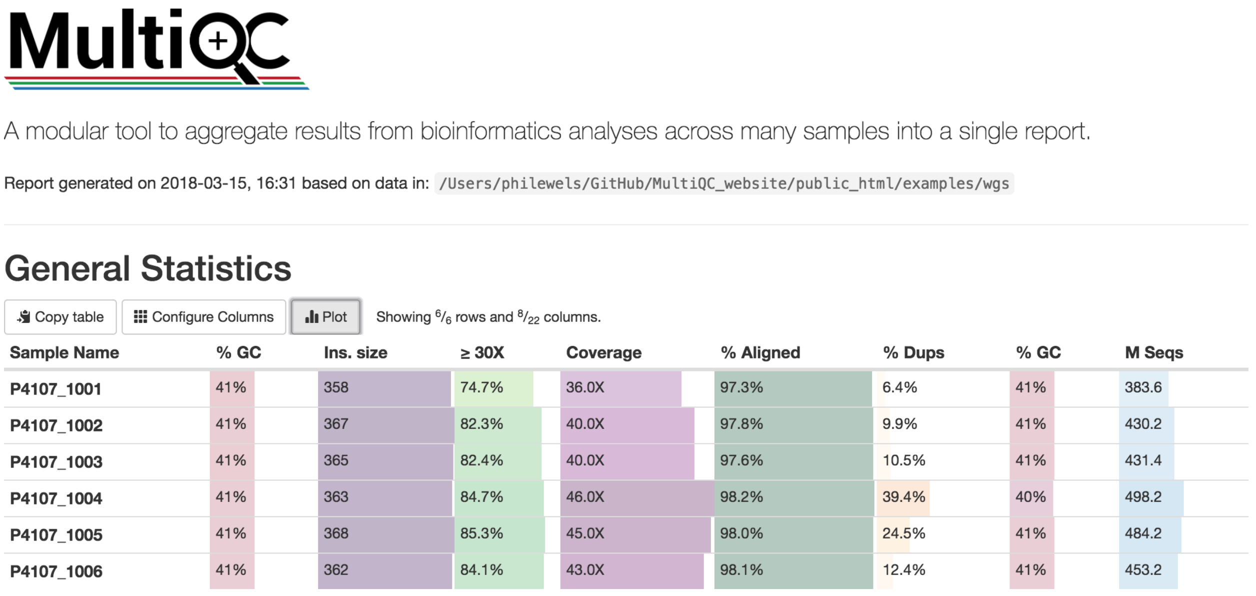


At this stage we have created different type of summary report using `FastQC` and `Qualimap`. To create a unique summary that integrate and compare all the generated reports, we will use `MultiQC`. If all the reports are in the same directory and its sub-directories, you can run `MultiQC` as follows:

```
multiqc .
```

A list of programs that generate output files recognized by `MultiQC` are availble here: https://github.com/ewels/MultiQC

Multiqc will create a summary report in `html` format that will let you compare all the summary reports for each of your samples:



MultiQC

A modular tool to aggregate results from bioinformatics analyses across many samples into a single report.

Report generated on 2018-03-15, 16:31 based on data in: `/Users/philewels/GitHub/MultiQC_website/public_html/examples/wgs`

General Statistics

Showing 6/6 rows and 8/22 columns.

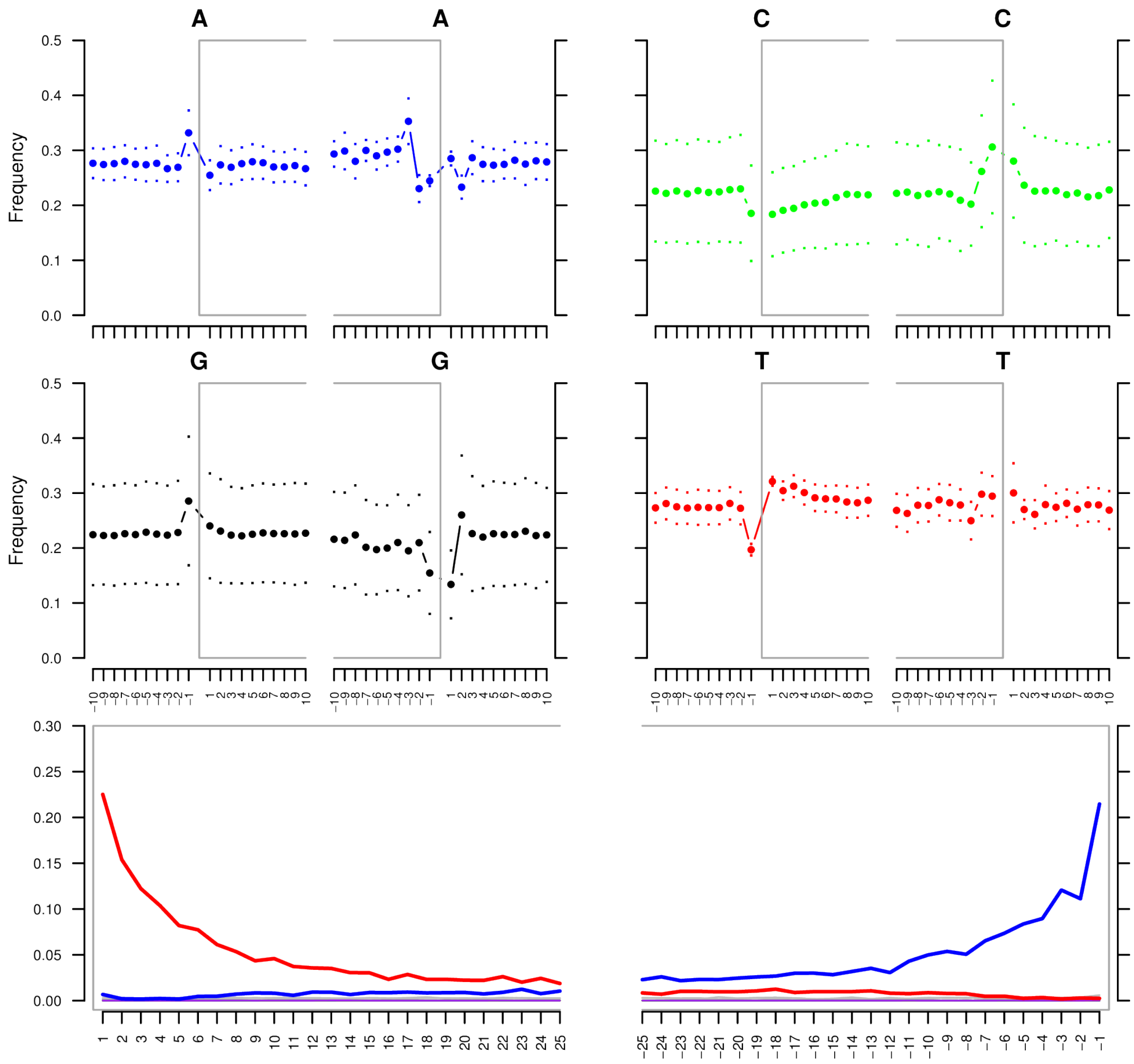
Sample Name	% GC	Ins. size	≥ 30X	Coverage	% Aligned	% Dups	% GC	M Seqs
P4107_1001	41%	358	74.7%	36.0X	97.3%	6.4%	41%	383.6
P4107_1002	41%	367	82.3%	40.0X	97.8%	9.9%	41%	430.2
P4107_1003	41%	365	82.4%	40.0X	97.6%	10.5%	41%	431.4
P4107_1004	41%	363	84.7%	46.0X	98.2%	39.4%	40%	498.2
P4107_1005	41%	368	85.3%	45.0X	98.0%	24.5%	41%	484.2
P4107_1006	41%	362	84.1%	43.0X	98.1%	12.4%	41%	453.2

1.4.4 Damage analysis and quality rescaling of the BAM file

To authenticate our analysis we will assess the *post-mortem* damage of the reads aligned to the reference sequence. We can track the *post-portem* damage accumulated by DNA molecules in the form of fragmentation due to depurination and cytosine deamination, which generates the typical pattern of **C->T** and **G->A** variation at the 5'- and 3'-end of the DNA molecules. To assess the *post-mortem* damage patterns in our `bam` file we will use `mapDamage`, which analyses the size distribution of the reads and the base composition of the genomic regions located up- and downstream of each read, generating various plots and summary tables. To start the analysis we need the final `bam` and the reference sequence:

```
mapDamage -i filename.final.sort.bam -r reference.fasta
```

`mapDamage` creates a new folder where the output files are created. One of these files, is named `Fragmisincorporation_plot.pdf` which contains the following plots:



If DNA damage is detected, we can run `mapDamage` again using the `--rescale-only` option and providing the path to the results folder that has been created by the program (option `-d`). This command will downscale the quality scores at positions likely affected by deamination according to their initial quality values, position in reads and damage patterns. A new rescaled `bam` file is then generated.

```
mapDamage -i filename.final.sort.bam -r reference.fasta --rescale-only -d results_
↪folder
```

You can also rescale the `bam` file directly in the first command with the option `--rescale`:

```
mapDamage -i filename.final.sort.bam -r reference.fasta --rescale
```

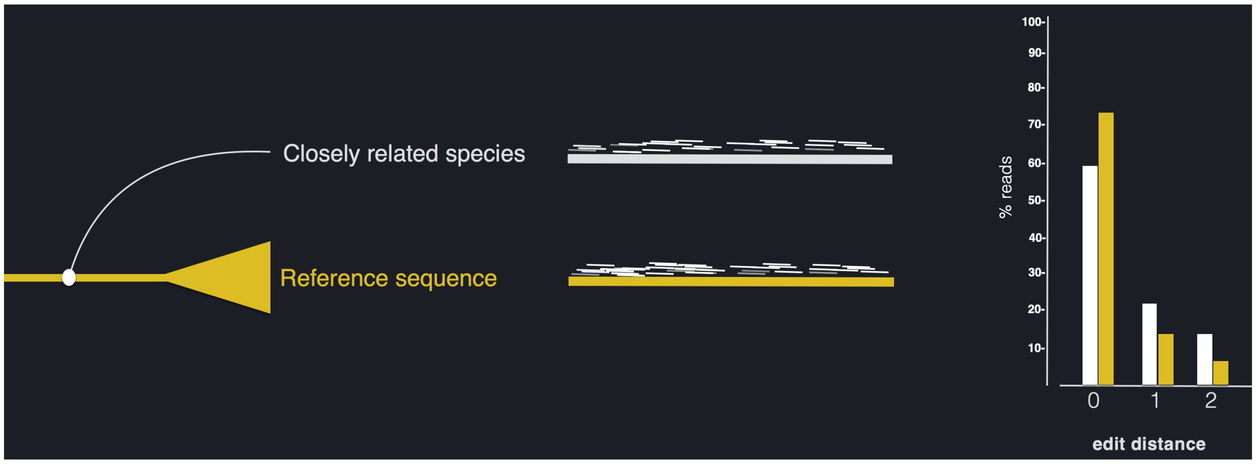
Note: Another useful tool for estimating *post-mortem* damage (PMD) is PMDTools. This program uses a model incorporating PMD, base quality scores and biological polymorphism to assign a PMD score to the reads. PMD > 0 indicates support for the sequence being genuinely ancient. PMDTools filters the damaged reads (based on the selected score) in a separate `bam` file which can be used for downstream analyses (e.g. variant call).

The rescaled `bam` file has to be indexed, as usual.

```
samtools index filename.final.sort.rescaled.bam
```

1.4.5 Edit Distance

The edit distance defines the number of nucleotide changes that have to be made to one read sequence for it to be identical to the reference sequence. To be more confident about the quality and authenticity of your sequencing data, you need to align your reads againt your reference sequence and the genome of a closely related species. Here we will align our `fastq` file against the Yersinia pseudotuberculosis genome, following all the steps from 4.1 to 4.4. The edit distance must be lower when aligning the reads to the reference sequence compared to the closely related species.



To calculate the edit distance we will use BAMStats, a tool for summarising Next Generation Sequencing alignments. The commands to generate summary-charts, including the edit distance is:

```
BAMStats -i filename.rescaled.bam -v html -d -q -o outfile.html
```

Option	Function
-i *filename*	SAM or BAM input file (must be sorted).
-v *html/simple*	View option for output format (currently accepts 'simple' or 'html'; default, simple).
-d	If selected, edit distance statistics will also be displayed as a separate table (optional).
-q	If selected, mapping quality (MAPQ) statistics will also be displayed as a separate table (optional).

1.5 Variant calling and visualization

Once the reads are aligned and the data authenticated through *post-mortem* damage analysis, we can analyse the variant positions in the samples against the reference sequence.

1.5.1 Variants calling

We will use two common tools for variants calling: Samtools, in particular `samtools mpileup`, in combination with `bcftools call` of the program BCFtools.

```
samtools mpileup -B -ugf reference.fasta filename.final.sort.rescaled.bam | bcftools␣
↪call -vmO z - > filename.vcf.gz
```

Samtools mpileup options	Function
-B, –no-BAQ	BAQ is the Phred-scaled probability of a read base being misaligned. Applying this option greatly helps to reduce false SNPs caused by misalignments.
-u, –uncompressed	Generate uncompressed VCF/BCF output, which is preferred for piping.
-g, –BCF	Compute genotype likelihoods and output them in the binary call format (BCF). As of v1.0, this is BCF2 which is incompatible with the BCF1 format produced by previous (0.1.x) versions of samtools.
-f, –fasta-ref *file*	The faidx-indexed reference file in the FASTA format. The file can be optionally compressed by bgzip.

BCFtools call options	Function
-v, –variants-only	Output variant sites only.
-m, –multiallelic-caller	Alternative modelfor multiallelic and rare-variant calling designed to overcome known limitations in -c calling model (conflicts with -c)
-g, –BCF	Compute genotype likelihoods and output them in the binary call format (BCF). As of v1.0, this is BCF2 which is incompatible with the BCF1 format produced by previous (0.1.x) versions of samtools.
-O, –output-type *b|u|z|v*	Output compressed BCF (b), uncompressed BCF (u), compressed VCF (z), uncompressed VCF (v).

The detected genetic variants will be stored in the `vcf` file. The genetic variants can be filtered according to some criteria using BCFtools:

```
bcftools filter -O z -o filename.filtered.vcf -s LOWQUAL -i'%QUAL>19' filename.vcf.gz
```

BCFtools filter options	Function
-O, –output-type *b|u|z|v*	Output compressed BCF (b), uncompressed BCF (u), compressed VCF (z), uncompressed VCF (v).
-o, –output *file*	Output file.
-s, –soft-filter *string|+*	Annotate FILTER column with <string> or, with +, a unique filter name generated by the program (“Filter%d”).
-i, –include *expression*	Include only sites for which expression is true.

Note: other options can be added when using BCFtools filter:

Option	Function
-g, –SnpGap *int*	Filter SNPs within *int* base pairs of an indel
-G, –IndelGap *int*	Filter clusters of indels separated by *int* or fewer base pairs allowing only one to pass

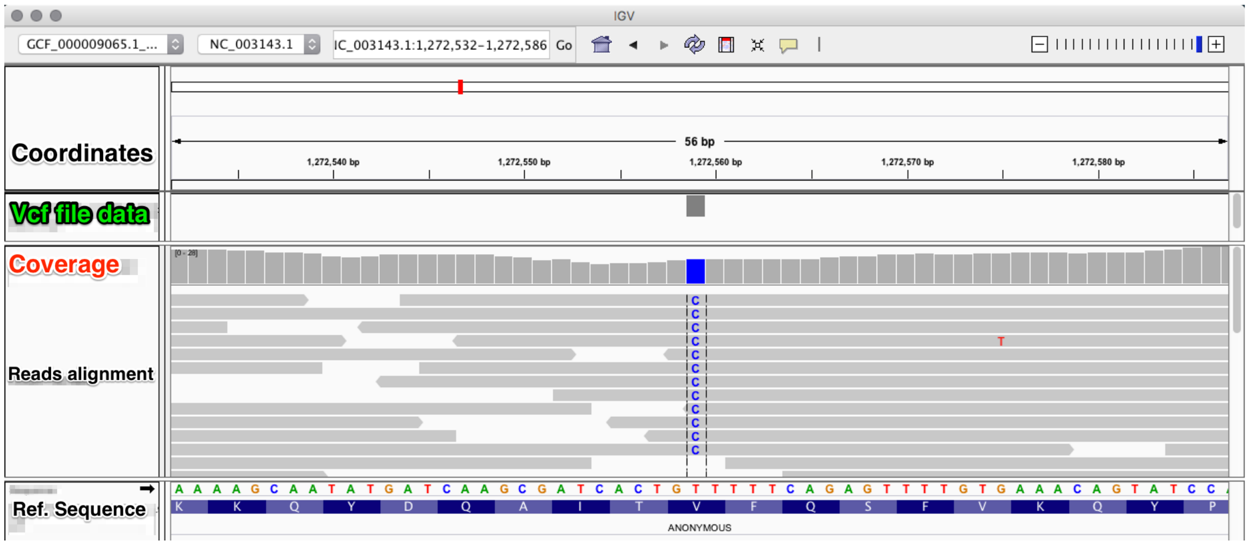
Instead of `samtools mpileup` and `bcftools call` (or in addition to) we can use `gatk HaplotypeCaller`:

```
java -jar GenomeAnalysisTK.jar -T HaplotypeCaller -R reference.fasta -I filename.
↪final.sort.rescaled.bam -o original.vcf.gz
java -jar GenomeAnalysisTK.jar -T VariantFiltration -R reference.fasta -V filename.
↪vcf.gz -o filename.filtered.vcf.gz --filterName 'Cov3|Qual20' --filterExpression
↪'DP>2||QUAL>19'
```

Now that you have your `vcf` file, you can open the file (use `nano` or `vim` in the server, or download the file in your laptop with `scp` and open it in a text editor) and try to search diagnostic variants (e.g. for classification). You can also visualize the variants in a specific program, as described below.

1.5.2 Variants visualization

To be able to visualize the variants in the `vcf` files, you can use the program `IGV`, which accepts multiple input files formats eg. `fasta`, `bam`, `vcf` and `gff`. After loading your `bam` file(s) and the corrsponding `vcf` file(s), you will see something likt that:


IGV
GCF_000009065.1_...
NC_003143.1
IC_003143.1:1,272,532-1,272,586
Go
Coordinates
56 bp
1,272,540 bp
1,272,550 bp
1,272,560 bp
1,272,570 bp
1,272,580 bp
Vcf file data
Coverage
Reads alignment
Ref. Sequence
A A A A G C A A T A T G A T C A A G C G A T C A C T G T T T T T C A G A G T T T T G T G A A A C A G T A T C C
K K Q Y D Q A I T V F Q S F V K Q Y P
ANONYMOUS

In this figure, we observe in the `bam` alignment file a T->C transition in the corresponding position.

1.6 Filtering, annotating and combining SNPs

To investigate the genetic variants in the `vcf` files we will use the program snpToolkit. The `-h` option will display the following message:

```
snpToolkit -h

positional arguments:
  {annotate,combine}   commands
    annotate           Please provide one or multiple vcf files
    combine            combine snpToolkit output files in one alignment in fasta format
```

Two options are possible: `annotate` or `combine`.

1.6.1 SNPs filtering and annotion

The `snpToolkit annotate` command will display general information about the usage of the program:

```
snpToolkit annotate

usage: snpToolkit annotate [-h] -i IDENTIFIER -g GENBANK
                              [-f EXCLUDECLOSESNPS] [-q QUALITY] [-d DEPTH]
```

(continues on next page)

(continued from previous page)

```
                                           [-r RATIO] [-e EXCLUDE] [--plot]

optional arguments:
  -h, --help            show this help message and exit

snpToolkit annotate required options:
  -i IDENTIFIER         provide a specific identifier to recognize the file(s)
                        to be analyzed
  -g GENBANK            Please provide a genbank file

snpToolkit annotate additional options:
  -f EXCLUDECLOSESNPS   exclude SNPs if the distance between them is lower then
                        the specified window size in bp
  -q QUALITY            quality score to consider as a cutoff for variant
                        calling. default value [20]
  -d DEPTH              minimum depth caverage. default value [3]
  -r RATIO              minimum ratio that correspond to the number of reads
                        that has the mutated allele / total depth in that
                        particular position. default value [0]
  -e EXCLUDE            provide a tab file with genomic regions to exclude in
                        this format: region1 start stop
```

Here is a simple example on how to use snpToolkit:

```
snpToolkit annotate -i VCF-filename.vcf.gz -g genbankFile.gbff -q 30 -d 5 -r 0.9
```

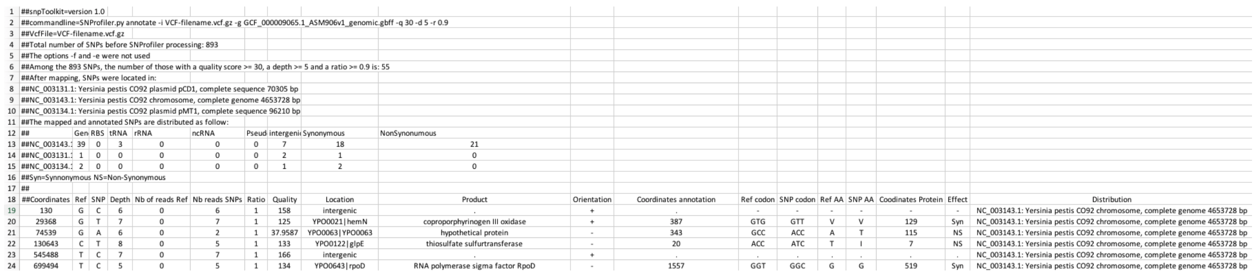
snpToolkit can automatically recogninze `vcf` files generated with the following programs: `samtools mpileup`, `gatk HaplotyCaller` and `freeBayes`. The `vcf` files could be gzipped or not. In the command line above, snpToolkit will filter and annotate all SNPs in the `vcf` file(s) that fullfil the following criteria: `quality >= 30`, `depth of coverage >= 5` and `ratio >= 0.9`.

Note: For each SNP position, the ratio (r) is calculated as follows:

r= dm / (dr + dm)

- dr= Number of reads having the reference allele
- dm= Number of reads having the mutated allele

The output file(s) of snpToolkit is a tabulated file(s) that you can open with Microsoft Excel and it will look as follow:



1	##snpToolkit=version 1.0																		
2	##commandline=SNProfiler.py annotate -i VCF-filename.vcf.gz -g GCF_000009065.1_ASM906v1_genomic.gbff -q 30 -d 5 -r 0.9																		
3	##VcfFile=VCF-filename.vcf.gz																		
4	##Total number of SNPs before SNProfiler processing: 893																		
5	##The options -f and -e were not used																		
6	##Among the 893 SNPs, the number of those with a quality score >= 30, a depth >= 5 and a ratio >= 0.9 is: 55																		
7	##After mapping, SNPs were located in:																		
8	##NC_003131.1: Yersinia pestis CO92 plasmid pCD1, complete sequence 70305 bp																		
9	##NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp																		
10	##NC_003134.1: Yersinia pestis CO92 plasmid pMT1, complete sequence 96210 bp																		
11	##The mapped and annotated SNPs are distributed as follow:																		
12	##	Gen	RBS	tRNA	rRNA	ncRNA	Pseud	intergeni	Synonymous	NonSynonumous									
13	##NC_003143.1	39	0	3	0	0	0	7	18	21									
14	##NC_003131.1	1	0	0	0	0	0	2	1	0									
15	##NC_003134.1	2	0	0	0	0	0	1	2	0									
16	##Syn=Synnonymous NS=Non-Synonymous																		
17	##																		
18	##Coordinates	Ref	SNP	Depth	Nb of reads Ref	Nb reads SNPs	Ratio	Quality	Location	Product	Orientation	Coordinates annotation	Ref codon	SNP codon	Ref AA	SNP AA	Coodinates Protein	Effect	Distribution
19	130	G	C	6	0	6	1	158	intergenic	.	+	.	-	-	-	-	-	-	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp
20	29368	G	T	7	0	7	1	125	YPO0021|hemN	coproporphyrinogen III oxidase	+	387	GTG	GTT	V	V	129	Syn	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp
21	74539	G	A	6	0	2	1	37.9587	YPO0063|YPO0063	hypothetical protein	-	343	GCC	ACC	A	T	115	NS	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp
22	130643	C	T	8	0	5	1	133	YPO0122|glpE	thiosulfate sulfurtransferase	-	20	ACC	ATC	T	I	7	NS	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp
23	545488	T	C	7	0	7	1	166	intergenic	.	+	.	.	.	.	.	.	.	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp
24	699494	T	C	5	0	5	1	134	YPO0643|rpoD	RNA polymerase sigma factor RpoD	-	1557	GGT	GGC	G	G	519	Syn	NC_003143.1: Yersinia pestis CO92 chromosome, complete genome 4653728 bp

The header of the generated snpToolkit output file includes useful information e.g. raw number of SNPs, Number of filtered SNPs, SNPs distribution, etc. . .

1.6.2 Compare and combine multiple annotation files

After generating a set of output files, you can run `snpToolkit combine`:

```
usage: snpToolkit combine [-h] --location LOCATION [-r RATIO] [-d DEPTH]
                             [--bam BAMFOLDER] [--snps {ns,s,all,inter}]

optional arguments:
  -h, --help            show this help message and exit

snpToolkit combine required options:
  --location LOCATION         provide the name of the locus you want to create
                                fasta alignment for

snpToolkit additional options:
  -r RATIO              SNP ratio
  -d DEPTH              depth cutoff for cheking missing data
  --bam BAMFOLDER       path to the folder containing bam files
  --snps {ns,s,all,inter}
                        Specify if you want to concatenate all SNPs or just
                        synonymous (s), non-synonymous (ns) or intergenic
                        (inter) SNPs. default [all]
```

`snpToolkit combine` will compare all the SNPs identified in each file and create two additional output files:

1) a tabulated files with all polymorphic sites
2) a `fasta` file.

As we will be working with ancient DNA, a small fraction of your genome could be covered. In this case we will use the option `--bam` to indicate the path to the folder containing the `bam` files. The option `-d` must be used with the option `--bam`. By default, all SNPs will be reported. This behaviour can be changed using the option `--snp`.

Note: It is also possible to use the option `--bam` with modern data as some genomic regions could be deleted.

The file reporting the polymorphic sites is organized as follows:

ID	Coordinates	REF	SNP	Columns with SNP information	sample1	sample2	sample3	sample4
snp1	130	A	T		1	1	1	1
snp2	855	C	G		0	0	?	1
snp3	1315	A	C		1	1	0	0
snp4	12086	G	A		1	0	?	0

The table above reports the distribution of all polymorphic sites in all provided files. As we provided the `bam` files of the ancient DNA samples, snpToolkit will check if the polymorphic sites (snp2 and snp4) are absent in sample3 because there is no SNP in that positions or because the region where the snps are located is not covered. In the latter case, snpToolkit will add a question mark `?` that reflects a missing data. From the table above, it will be possible to generate a `fasta` file, like the one below:

```
>Reference
ATCGGGTATGCCAATGCGT
>Sample1
ACCGGGTATGCCAATGTGT
>Sample2
```

(continues on next page)

(continued from previous page)

```
ATTGGGTATGCCAGTGCGT
>Sample3
?TTGAGT?TGTCA?TACGT
>Sample4
ATCGGGTATGCCAATGCGT
```

The `fasta` output file will be used to generate a maximum likelihood tree using `IQ-TREE`

1.6.3 Phylogenetic tree reconstruction

There are several tools to build phylogenetic trees. All of these tools, use an alignment file as input file. Now that we have generated an alignment file in `fasta` format, we will use `IQ-TREE` to build a maximum likelihood tree. We use `IQ-TREE` for several reasons:

- It performs a composition chi-square test for every sequence in the alignment. A sequence is denoted failed if its character composition significantly deviates from the average composition of the alignment.
- Availability of a wide variety of phylogenetic models. `IQ-TREE` uses ModelFinder to find the best substitution model that will be used directly to build the maximum likelihood phylogenetic tree.
- Multithreading

The phylogenetic tree generated can be visualized using `Figtree`.

1.7 DO-IT-YOURSELF

In this final hands-on session you will analyse shotgun (reduced) sequencing data generated from an ancient human tooth.The genomic library built from the DNA extract was sequenced on an Illumina platform in paired-end mode. Your task is:

1. Process the raw reads (remove adapters, merge the reads, section 2).
2. Align the reads to the human mitochondrial DNA (mtDNA) reference sequence, assess the damage of DNA molecules, call the variants (sections 4-5-6).
3. Run the metagenomic screning of the DNA extract with Kraken using the Minikraken database (section 3).

After reads pre-processing it is up to you whether first aliging the reads or screening the metagenomic content.

- **Option 1**: You can use your `vcf` file to assign an haplogroup to the human samples that you analysed. Some useful tools for haplogroup assignation:
  - Check the variant positions in Phylotree (http://www.phylotree.org/)
  - Load the `vcf` file in Haplogrep (https://haplogrep.uibk.ac.at)
- **Option 2**: Run again the metagenomic screening with a Custom Database of Kraken (provided by us), and compare the results with those obtained with Minikraken.

1.8 R session

In this hands-on we will use R to run analyses and create charts from the abundance tables that we generated with Kraken2.

First of all, in R, we must set up the folder which contains the abundance table:

```
setwd("~/Documents/WORK/2-PROJECTS/HiddenFoods/HiddenFoods_analysis/krk2_genome_
↪length_normalized_abundances_July2019/species")
```

Then we can import the abundance table files:

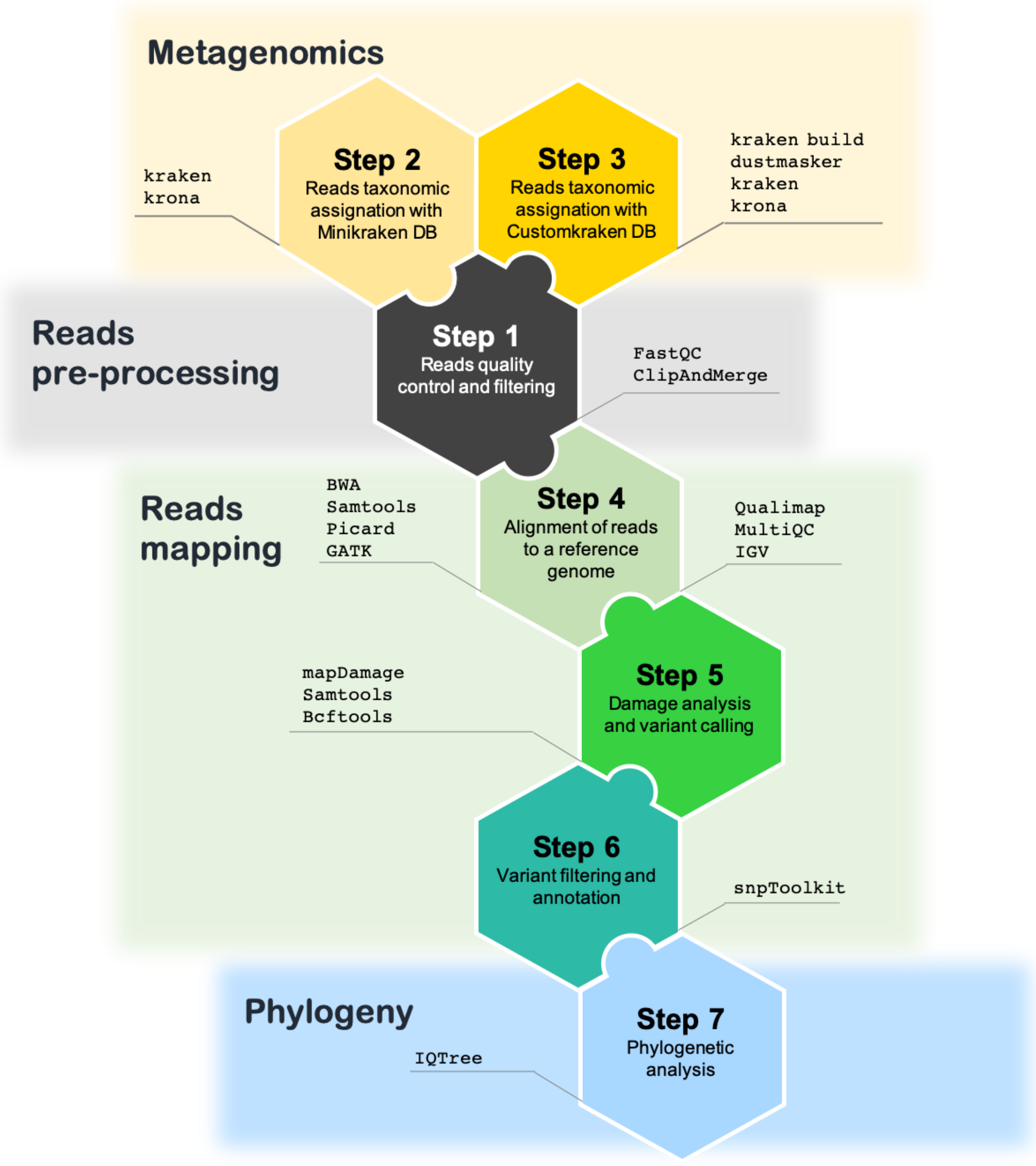
```
# --> you can edit the files, eg remove last column 'Detail!
# dataframes in R must do not allow identical row names, force row names as numbering␣
↪by using row.names=NULL.
#EOF issue with single quotes (') in species names, use read.delim
HiddenFoods_Jan2019.krk2.norm.species = read.delim("HiddenFoods_Jan2019.krk2.rcf.
↪abundance.species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
HiddenFoods_May2019.krk2.norm.species = read.delim("HiddenFoods_May2019.krk2.rcf.
↪abundance.species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Warinner2014.krk2.norm.species = read.delim("Warinner2014.krk2.rcf.abundance.species.
↪norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Velsko_modern_calculus.krk2.norm.species = read.delim("Velsko_modern_calculus.krk2.
↪rcf.abundance.species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Velsko_ancient_calculus_total.krk2.norm.species = read.delim("Velsko_ancient_calculus_
↪total.krk2.rcf.abundance.species.norm2.final", header=T, fill=T, row.names=NULL,␣
↪sep="\t")
Plaque.krk2.norm.species = read.delim("Plaque.krk2.rcf.abundance.species.norm2.final",
↪ header=T, fill=T, row.names=NULL, sep="\t")
Mann2018.krk2.norm.species = read.delim("Mann2018.krk2.rcf.abundance.species.norm2.
↪final", header=T, fill=T, row.names=NULL, sep="\t")
Baboons_Ozga.krk2.norm.species = read.delim("Baboons_Ozga.krk2.rcf.abundance.species.
↪norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Weyrich2017.krk2.norm.species = read.delim("Weyrich2017.krk2.rcf.abundance.species.
↪norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Baboons_Egypt_all.krk2.norm.species = read.delim("Baboons_Egypt_all.krk2.rcf.
↪abundance.species.flt10.teeth.env.norm2.final", header=T, fill=T, row.names=NULL,␣
↪sep="\t")
Baboons_Egypt_nofilter.krk2.norm.species = read.delim("Baboons_Egypt_nofilter.krk2.
↪rcf.abundance.species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Baboons_Egypt_teeth.krk2.norm.species = read.delim("Baboons_Egypt_teeth.krk2.rcf.
↪abundance.species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Skin.krk2.norm.species = read.delim("Skin.krk2.rcf.abundance.species.norm2.final",␣
↪header=T, fill=T, row.names=NULL, sep="\t")
ObregonTito_gut.krk2.norm.species = read.delim("ObregonTito_gut.krk2.rcf.abundance.
↪species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Soil.krk2.norm.species = read.delim("Soil.krk2.rcf.abundance.species.norm2.final",␣
↪header=T, fill=T, row.names=NULL, sep="\t")
Chimps.krk2.norm.species = read.delim("Chimps_Ozga.krk2.rcf.abundance.species.norm2.
↪final", header=T, fill=T, row.names=NULL, sep="\t")
Brealey_animals.krk2.norm.species = read.delim("Brealey_animals.krk2.rcf.abundance.
↪species.norm2.final", header=T, fill=T, row.names=NULL, sep="\t")
Brealey_animals.krk2.norm.species = Brealey_animals.krk2.norm.species[,-c(2:5,7,8)]  ␣
↪         #remove Blanks from filtrated Brealey, Gb1reg/flt, Gb2reg/flt
Eisenhofer_Japan_flt.krk2.norm.species = read.table("Eisenhofer_Japan.krk2.rcf.
↪abundance.species.norm2.final.flt", header=T, fill=T, row.names=NULL, sep="\t")
Eisenhofer_Japan_flt.krk2.norm.species = Eisenhofer_Japan_flt.krk2.norm.species[,-
↪c(19,20)]   #remove EBC-2 in  Eisenhofer which has no classified species, and␣
↪Details column.
```

1.8.1 Barplot

1.8.2 UPGMA

1.8.3 nMDS

Course overview



- genindex

- modindex
- search