# Practical: Assembling a SARS-Cov-2 genome step by step

### Overview

*Introduction*

* *Input data*
* *Software*
* *How to use this document*
* *Some general advice*

*Step 1: Map your data to a reference genome*

*Step 2: Convert from “sam” to “bam” and sort*

*Step 3: Trim the primer-sequences*

*Step 4: Generate your genome sequence*

*Step 5: Run nextclade to determine SARS-Cov-2 lineage*

The overall purpose of this practical is to provide an understanding of the processing steps involved in generating a SARS-CoV-2 genome, using a reference genome. Therefore, each step will be executed manually (we will show you how the steps can be automated in a later session). The practical requires both some raw data and several types of software, as detailed here below.

#### Input data

In order to generate a SARS-CoV-2 genome, you will need some sequencing data. In this practical, we are using Illumina paired-end reads (2 x 75bp), which in turn were generated from PCR-enriched viral cDNA. Specifically, we used a modified version of the artic-V3 primer-scheme, with “spike-in” primers to counteract various mutations (see “Primerscheme\_table\_updates.pdf” for more details).

An example dataset using this set-up can be downloaded from ENA:

https://www.ebi.ac.uk/ena/browser/view/ERR11759656

You will also need a reference genome, to map your read against (as this is a reference-based genome assembly), and a bed-file with the positions of primers relative to your reference genome. The bed-file corresponding to the modified artic-V3 primer-scheme and the matching reference genome is provided here on the course site:

nCoV-2019\_v3\_ssi\_v5.primer.bed

SARS-CoV-2.reference.fasta

#### Software

Several different software packages are required in order to run the commands in this practical. If you are using linux, and have conda installed, you can use the yaml-file uploaded here on the course site:

conda env create -f env\_BTG\_covid\_assembly.yml

conda activate env\_BTG\_covid\_assembly

#### How to use this document

Software commands are highlighted in grey. Arguments must be filled in manually (so don’t copy-paste the commands!); all arguments are indicated by square brackets. For example, if you need to use the following command:

bgzip -k [input vcf-file]

on a vcf-file named “exciting.vcf”, you would type the following in your terminal:

bgzip -k exciting.vcf

All lines starting with the hash-tag symbol “#” are comments. These are there to help you understand the code, and should not be executed.

#### Some general advice

In this practical, you will generate a SARS-CoV-2 genome, step by step (without using a pipeline). You will generate a **lot** of intermediate files. It is therefore important to keep your working directory and files well organized, and check the output of your commands:

* Name your output files carefully. Pick file-names that provide an idea about what they contain
* Keep an eye on where you output-files are going. Use the list command (ls) excessively
* Consider whether it might be helpful to create some sub-folders
* Look inside your files (less, head, tail..). A file can be empty!

### Step 1: Map your data to a reference genome

**Note**: in the interest of saving time, we have already checked the quality of the raw-data for you. Otherwise, you should of course also do that.

The first step of a reference-based genome assembly is to map your reads to your reference genome.

* Do you think the choice of reference-genome has an impact? If so, how?

##Map your reads to the reference genome

#Check the usage info for “bwa” (just type the tool name). How many alignment algorithm options does “bwa” have?

bwa

#Index the reference genome

bwa index [reference genome]

#run “bwa mem”

bwa mem [reference genome] [input fastq-file R1] [input fastq-file R2] -o [out.sam]

* You should have generated a “SAM file” now. Take a quick look inside your sam-file (less, head, tail..). What are we looking at here? (hint: google “sam file format”).

### Step 2: Convert from “sam” to “bam”, and sort

Have you noticed the size of that sam-file? Sam-files can get enormous, especially if you have a lot of reads and a large genome. Therefore, it is good practice to convert your sam-files to the “binary” equivalent, which is much smaller (whats a binary file? Google!). This can be done with “samtools”.

##Generate a bam-file from your sam-file

#Check usage info for ”samtools view”. What does the ”-b” flag mean?

samtools view

#Convert “sam-file” to “bam-file”, with “samtools”

samtools view -b [input sam-file] -o [output bam-file]

* How big is your bam-file, compared to your sam-file? What happens if you try to look inside it?

Fortunately, there is a tool for looking inside bam-files in the “samtools” package:

#Use “samtools view” to look inside your bam-file

samtools view [input bam-file] | less

Did you notice the “pipe” in the previous command (vertical line)? You can use any of the bash-commands that you would normally use on a text-file with a piped command (less, head, tail, wc). You can even use multiple “samtools” commands one after another. In fact, its very common to pipe “samtools” commands together, to avoid a lot of large intermediate files.

Lets try to convert your sam-file to bam-file, and sort the alignments according to their mapping position, in **one single command.**

**Note**: make sure to give your bamfile a new name! Perhaps something like “[sample-id]\_sorted.bam”?.

#Piped samtools command (sam-to-bam and sort)

samtools view -b [input sam-file] | samtools sort -o [output bam-file]

You should now have a sorted bam-file. Take a look at the top of the sorted bam-file, and compare to the unsorted bam-file:

#Check top of bam-files with samtools

samtools view [unsorted bamfile] | head -n5

samtools view [sorted bamfile] | head -n5

* See the difference? Hint: check column 4

Sorting bam-files is a common task, since most software working on bam-files can’t handle unsorted bam-files (including the ones we will be using in this practical).

### Step 3: trim the primer-sequences

When using a PCR-enrichment strategy for genome sequencing, the primer-sequences will be sequenced along with the genome, since they are part of the PCR products. If the primers match the reference genome, it doesn’t really matter. But if the primers don’t match perfectly (but are still able to generate PCR products), “mutations” are effectively introduced into the data. Therefore, reads that are likely to come from a primer should be “masked” (i.e. labelled in the bam-file, so downstream software can ignore them).

In the SARS-CoV-2 pipeline used for routine surveillance at SSI, we use a tool called “iVar” for this purpose. To use this tool, you need to give it a “bed-file”, specifying where your primers are located (whats a “bed-file”? Google is your friend!). Keep an eye on your terminal while “ivar” is running, this tool is quite verbose.

**Note**: make sure to give your primer-trimmed bamfile a new name! Perhaps something like “[sample-id]\_primertrimmed.bam”?

##Run primer-trimming on bam-file, with “ivar”

#Check out the “ivar” usage info

ivar

#Check out the “ivar trim” usage info

ivar trim

#Run “ivar trim” on your sorted bam-file

ivar trim -e -i [input sorted bamfile] -b [input bed-file] -m 30 -q 20 -p [prefix for the primer-trimmed bam-file]

* How many reads got trimmed according to ivar?

### Step 4: Generate your genome sequence

Having trimmed the primer-sequences, we are ready to generate our new genome sequence from the bam-file. There are many possible software choices for this task. In the SARS-CoV-2 pipeline currently used for routine surveillance at SSI, we use “bcftools”.

Similarly to “samtools”, “bcftools” is another collection of tools, which was designed for working with “variant” files (i.e. “vcf-files”, and the binary counter-part “bcf-files”). As for “iVar”, we need to sort the primer-trimmed “bam-file” before it can be processed.

#Sort your primer-trimmed bam-file

samtools sort [primer-trimmed bam-file] -o [output bam-file]

The genome sequence is generated in two steps:

1. Generate a “vcf-file” detailing the differences relative to the reference genome (vcf=”Variant Call Format”)
2. Generate a “fasta-file” from the “vcf-file”.

Our command for generating the vcf-file is rather complex, it contains a lot of pipes. We have split the command into multiple lines for better readability here (one per pipe segment), but it should be **one single line** in your terminal. This is the command (don’t execute it just yet!):

#Generate your vcf-file with a long piped command

bcftools mpileup -B -d 250 --max-idepth 1000 --annotate INFO/AD,FORMAT/AD -Q 30 -f [reference genome] [primer-trimmed bamfile] |

bcftools call -Ou -m |

bcftools +fill-tags -- -t FORMAT/VAF |

bcftools +setGT -- -t q -i 'GT="1/1" && FORMAT/VAF < 0.8' -n 'c:0/1' |

bcftools +setGT -- -t q -i 'GT="0/1" && FORMAT/VAF >= 0.8' -n 'c:1/1' |

bcftools filter -o [output vcf-file] -e 'INFO/IMF < 0.5' --

Briefly, what is happening is the following:

1. “bcftools mpileup” collects information about mapped reads on each position in the reference genome, using the provided bam-file and reference genome file. In case of high depth, only a subset of the mapped reads is used (depth cut-off). There is also a quality threshold for the read alignment (reads mapped with low score are ignored). Moreover, we are adding some extra information to the vcf-file, using the “--annotate” flag, namely the “allele depth” for detected variants.
2. We are using “bcftools +setGT” to set a custom “genotype” value in the “genotype” field. This is where we decide whether a base-call should be considered ambiguous or not. Here, we have specified that at least 80% of the reads mapping to a position should agree, otherwise the genotype will be set to “0/1” (which will be interpreted as ambiguous downstream)
3. We are using “bcftools filter” to remove deletions from the vcf-file if they have low support (here set to less than 50% of the mapped reads supporting it)

Now, construct your command in a text-file, and copy it into the terminal when you are ready to execute. Be particularly careful with the quotation marks!

If you managed to write the whole command correctly, you should now have a vcf-file in your current directory. Take a look inside it.

As for sam-files, vcf-files can also get very large. It is therefore good practice to compress the file. Moreover, like “bwa”, “bcftools” also require that the vcf-file is indexed, for downstream processing. Lets fix that:

#Compress the vcf-file

bgzip -k [input vcf-file]

#Index the compressed vcf-file

tabix -p vcf [input compressed vcf-file]

**Note**: here, we told “bgzip” to keep the original vcf-file (using the “-k” flag). Normally, we wouldn’t do that since the point is to save space. However, while you are learning about vcf-files, it is nice to have the option to look inside the uncompressed file.

Now, finally! Generate your genome-sequence:

#Generate your genome sequence with bcftools

bcftools consensus -a 'N' -p [desired prefix for fasta-header] -f [reference genome file] -H I -i 'INFO/DP >= 10' [input compressed and indexed vcf-file] > [output genome file-name]

### Step 5: Run nextclade to determine SARS-Cov-2 lineage

The specific lineage of SARS-Cov-2 for your sample can be determined with the tool “nextclade”. In brief, nextclade makes a lineage call by assigning your genome sequence to a clade in a phylogenetic tree composed from a representative data-set of available genomes (more about phylogenies on Day 6). To run nextclade, you will therefore need to provide reference data-set, which can be downloaded with nextclade as detailed here below.

##Run nextclade on your fasta-file (genome-sequence)

#Check the usage info for “nextclade”

nextclade -h

#Get a list of available reference data-sets

nextclade dataset list

#Download the SARS-CoV-2 dataset containing the Wuhan reference genome

nextclade dataset get --name sars-cov-2 --output-zip sars-cov-2.zip

unzip sars-cov-2.zip

#Check the usage info for “nextclade run”

nextclade run -h

#Run nextclade

nextclade run -D [/path/to/nextclade/reference\_dataset/folder] [input fasta-file]

--output-tsv [outfile-name]

Take a look at the output-file, which lineage does your SARS-Cov-2 sample belong to?

There is a lot of additional information in the file, including QC-scores for the lineage assignment, substitutions, frameshifts etc. If we have time, we will take a closer look at this file in the coding session later today.

# Bonus exercises

**Bonus exercise 1:** There is a large collection of tools developed specifically for “sam-files”, called “samtools”. Try the following commands out.

#Check the “samtools” usage info

samtools

#Check the “samtools flagstat” usage info

samtools flagstat

#run “samtools flagstat” to get some summary info for your sam-file (output gets printed in the terminal)

samtools flagstat [input samfile]

* Are there any unmapped reads?
* Does any of the numbers in the summary-info correspond to the number of lines in the file?

**Bonus-exercise 2**: What has changed in the primer-trimmed bamfile compared to the untrimmed bam-file (step 3)? Hint: check out the CIGAR field in the bam-file for a few reads mapping to a primer-region.

**Bonus exercise 3**: You got some verbose output while running the consensus-caller (re-run it, so you can see it again). Perhaps you got some warning messages about overlapping variants? What does that mean? Have a look inside your vcf-file!

# Acknowledgements:

This practical was written by Kirsten Ellegaard (SSI). The content is based on the current pipeline used for SARS-CoV-2 genome sequencing surveillance, which was developed by Raphael Sieber (SSI) and Thor Bech Johannesen (SSI).