de.NBI Nanopore Training Course Documentation

Release stable

Oct 05, 2022

Contents

CHAPTER 1

Best Practice and SARS-CoV-2 applications

Welcome to the three-day nanopore Hands-On training course. This tutorial will guide you through the typical steps of the analysis of ONT sequencing data ranging from basecalling, quality assessment over assembly to sequence polishing and finally assembly evaluation. A focus in this workshop is on the analysis of Nanopore sequenced SARS-CoV-2 genomes and a suitable workflow for resequencing based on the ARTIC bioinformatics protocol.

1.1 Connect to your virtual machine

The first thing you need to do is to connect to your virtual machine with the X2Go Client. If you are working with your laptop and haven't installed it yet - you can get it here: https://wiki.x2go.org/doku.php/download:start

Enter the IP of your virtual machine, the port (ssh port), the username "ubuntu" and select your ssh key. When you have successfully connected to your machine, open a terminal.

You can get all the information in the cloud portal under Overview->Instances. Then go to your virtual machine and click "How to connect". There is an ssh command like:

```
ssh -p PORT_NUMBER -i /path/to/your/ssh/private/key ubuntu@IP
```

This is the IP and Port-Number you need to enter in the X2Go Interface.

Start a terminal.

If you want to disable the annoying system beeps:

```
xset -b
```

1.1.1 Create a working directory

When you have successfully connected to your VM, let's create a working directory for the course. There is an ephemeral drive in /mnt/ with enough storage:

```
df -h /mnt/

Filesystem      Size  Used Avail Use% Mounted on
/dev/vdb        147G   61M  140G   1% /mnt
```

It is owned by root:

```
ls -ld /mnt/

drwxr-xr-x 3 root root 4096 Sep 20  2018 /mnt/
```

Change the owner to ubuntu, so you can write and read files in the directory:

```
sudo chown ubuntu:ubuntu /mnt/
```

And finally, create a link in your home directory to the mounted volume:

```
ln -s /mnt/ workdir
```

Enter the workdir:

```
cd ~/workdir
```

1.2 The Tutorial Data Sets

We have two tutorial datasets prepared for you. One is a SARS-Cov2 WGS dataset from a patient in Hongkong. We will use that sample for the assembly part of the tutorial. The dataset can also be found in the SRA: https://www.ncbi.nlm.nih.gov/sra/SRX8154258

The other one is our own amplicon based dataset created with ARTIC primers from an outbreak in Guetersloh/Germany. Since we don't have the raw data for the SRA dataset, we will start with the basecalling for our dataset.

The tutorial dataset is located in our object store. First create a data folder, and enter it:

```
cd ~/workdir
mkdir data_artic
cd data_artic
```

We have 5 different barcodes available. The datasets are highly reduced in order to achieve reasonable runtimes. Download one of them (use 1 of the commands below):

```
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪tiny_01.tar.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪tiny_02.tar.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪tiny_03.tar.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪tiny_04.tar.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪tiny_05.tar.gz
```

Then, unpack the tar archives:

```
ls | xargs -n 1 tar -xzvf
```

and remove the tar archives:

```
rm barcode_tiny_0*.tar.gz
```

and rename the directory, such that everyone of you has the same directory name:

```
mv ~/workdir/data_artic/barcode_tiny_<number> ~/workdir/data_artic/barcode_tiny/
```

Have a short look, on what is contained within the barcode directory:

```
ls -l ~/workdir/data_artic/barcode_tiny/
```

There is only one fast5 file contained in the tiny dataset, usually you have a number of fast5 files like this:

```
total 33703024
-rw-rw-r-- 1 ubuntu ubuntu 85000469 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_0.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 85265241 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_1.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 85383688 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_10.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 86318724 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_100.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 86630435 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_101.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 86527663 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_102.
↪fast5
-rw-rw-r-- 1 ubuntu ubuntu 86645812 Apr  9  2020 FAN22359_pass_barcode01_ca07d29b_103.
↪fast5
```

1.3 Basecalling

We will perform a basecalling of the raw data with guppy.

1.3.1 Basecalling with Guppy

Guppy is a data processing toolkit that contains the Oxford Nanopore Technologies' basecalling algorithms, and several bioinformatic post-processing features. It is provided as binaries to run on Windows, OS X and Linux platforms, as well as being integrated with MinKNOW, the Oxford Nanopore device control software.

Early downstream analysis components such as barcoding/demultiplexing, adapter trimming and alignment are contained within Guppy. Furthermore, Guppy now performs modified basecalling (5mC, 6mA and CpG) from the raw signal data, producing an additional FAST5 file of modified base probabilities.

The command we are using for basecalling with Guppy is:

```
guppy_basecaller_cpu
```

Let's have a look at the usage message for guppy_basecaller_cpu:

```
guppy_basecaller_cpu --help

: Guppy Basecalling Software, (C) Oxford Nanopore Technologies, Limited. Version 3.1.
↪5+781ed57

Usage:

With config file:
  guppy_basecaller_cpu -i <input path> -s <save path> -c <config file> [options]
With flowcell and kit name:
  guppy_basecaller_cpu -i <input path> -s <save path> --flowcell <flowcell name>
    --kit <kit name>
List supported flowcells and kits:
  guppy_basecaller_cpu --print_workflows
```

Beside the path of our fast5 files (-i), the basecaller requires an output path (-s) and a config file or the flowcell/kit combination. In order to get a list of possible flowcell/kit combinations and config files, we use:

```
guppy_basecaller_cpu --print_workflows

flowcell   kit         barcoding config_name
FLO-MIN110 SQK-CAS109           dna_r10_450bps_hac
FLO-MIN110 SQK-CS9109           dna_r10_450bps_hac
FLO-MIN110 SQK-DCS108           dna_r10_450bps_hac
FLO-MIN110 SQK-DCS109           dna_r10_450bps_hac
FLO-MIN110 SQK-LRK001           dna_r10_450bps_hac
FLO-MIN110 SQK-LSK108           dna_r10_450bps_hac
FLO-MIN110 SQK-LSK109           dna_r10_450bps_hac
FLO-MIN110 SQK-LSK109-XL          dna_r10_450bps_hac
FLO-MIN110 SQK-LSK110           dna_r10_450bps_hac
FLO-MIN110 SQK-LWP001           dna_r10_450bps_hac
FLO-MIN110 SQK-PCS108           dna_r10_450bps_hac
FLO-MIN110 SQK-PCS109           dna_r10_450bps_hac
FLO-MIN110 SQK-PRC109           dna_r10_450bps_hac
FLO-MIN110 SQK-PSK004           dna_r10_450bps_hac
FLO-MIN110 SQK-RAD002           dna_r10_450bps_hac
FLO-MIN110 SQK-RAD003           dna_r10_450bps_hac
FLO-MIN110 SQK-RAD004           dna_r10_450bps_hac
FLO-MIN110 SQK-RAS201           dna_r10_450bps_hac
FLO-MIN110 SQK-RLI001           dna_r10_450bps_hac
FLO-MIN110 VSK-VBK001           dna_r10_450bps_hac
FLO-MIN110 VSK-VSK001           dna_r10_450bps_hac
FLO-MIN110 VSK-VSK002           dna_r10_450bps_hac
FLO-MIN110 SQK-16S024 included  dna_r10_450bps_hac
FLO-MIN110 SQK-PCB109 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RBK001 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RBK004 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RLB001 included  dna_r10_450bps_hac
FLO-MIN110 SQK-LWB001 included  dna_r10_450bps_hac
FLO-MIN110 SQK-PBK004 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RAB201 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RAB204 included  dna_r10_450bps_hac
FLO-MIN110 SQK-RPB004 included  dna_r10_450bps_hac
FLO-MIN110 VSK-VMK001 included  dna_r10_450bps_hac
FLO-MIN110 VSK-VMK002 included  dna_r10_450bps_hac
FLO-PRO001 SQK-LSK109           dna_r9.4.1_450bps_hac_prom
FLO-PRO001 SQK-LSK109-XL          dna_r9.4.1_450bps_hac_prom
```

(continues on next page)

(continued from previous page)

```
FLO-PRO001 SQK-DCS109           dna_r9.4.1_450bps_hac_prom
FLO-PRO001 SQK-PCS109           dna_r9.4.1_450bps_hac_prom
FLO-PRO001 SQK-PRC109           dna_r9.4.1_450bps_hac_prom
FLO-PRO001 SQK-PCB109 included  dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-LSK109           dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-LSK109-XL          dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-DCS109           dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-PCS109           dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-PRC109           dna_r9.4.1_450bps_hac_prom
FLO-PRO002 SQK-PCB109 included  dna_r9.4.1_450bps_hac_prom
FLO-MIN111 SQK-CAS109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-CS9109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-DCS108           dna_r10.3_450bps_hac
FLO-MIN111 SQK-DCS109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-LRK001           dna_r10.3_450bps_hac
FLO-MIN111 SQK-LSK108           dna_r10.3_450bps_hac
FLO-MIN111 SQK-LSK109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-LSK109-XL          dna_r10.3_450bps_hac
FLO-MIN111 SQK-LSK110           dna_r10.3_450bps_hac
FLO-MIN111 SQK-LWP001           dna_r10.3_450bps_hac
FLO-MIN111 SQK-PCS108           dna_r10.3_450bps_hac
FLO-MIN111 SQK-PCS109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-PRC109           dna_r10.3_450bps_hac
FLO-MIN111 SQK-PSK004           dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAD002           dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAD003           dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAD004           dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAS201           dna_r10.3_450bps_hac
FLO-MIN111 SQK-RLI001           dna_r10.3_450bps_hac
FLO-MIN111 VSK-VBK001           dna_r10.3_450bps_hac
FLO-MIN111 VSK-VSK001           dna_r10.3_450bps_hac
FLO-MIN111 VSK-VSK002           dna_r10.3_450bps_hac
FLO-MIN111 SQK-16S024 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-PCB109 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RBK001 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RBK004 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RLB001 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-LWB001 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-PBK004 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAB201 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RAB204 included  dna_r10.3_450bps_hac
FLO-MIN111 SQK-RPB004 included  dna_r10.3_450bps_hac
FLO-MIN111 VSK-VMK001 included  dna_r10.3_450bps_hac
FLO-MIN111 VSK-VMK002 included  dna_r10.3_450bps_hac
FLO-FLG001 SQK-CAS109           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-CS9109           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-DCS108           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-DCS109           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LRK001           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LSK108           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LSK109           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LSK109-XL          dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LSK110           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LWP001           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-PCS108           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-PCS109           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-PRC109           dna_r9.4.1_450bps_hac
```

(continues on next page)

(continued from previous page)

```
FLO-FLG001 SQK-PSK004           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAD002           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAD003           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAD004           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAS201           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RLI001           dna_r9.4.1_450bps_hac
FLO-FLG001 VSK-VBK001           dna_r9.4.1_450bps_hac
FLO-FLG001 VSK-VSK001           dna_r9.4.1_450bps_hac
FLO-FLG001 VSK-VSK002           dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-16S024 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-PCB109 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RBK001 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RBK004 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RLB001 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-LWB001 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-PBK004 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAB201 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RAB204 included  dna_r9.4.1_450bps_hac
FLO-FLG001 SQK-RPB004 included  dna_r9.4.1_450bps_hac
FLO-FLG001 VSK-VMK001 included  dna_r9.4.1_450bps_hac
FLO-FLG001 VSK-VMK002 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-CAS109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-CS9109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-DCS108           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-DCS109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LRK001           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LSK108           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LSK109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LSK109-XL          dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LSK110           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LWP001           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PCS108           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PCS109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PRC109           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PSK004           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAD002           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAD003           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAD004           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAS201           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RLI001           dna_r9.4.1_450bps_hac
FLO-MIN106 VSK-VBK001           dna_r9.4.1_450bps_hac
FLO-MIN106 VSK-VSK001           dna_r9.4.1_450bps_hac
FLO-MIN106 VSK-VSK002           dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-16S024 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PCB109 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RBK001 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RBK004 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RLB001 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-LWB001 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-PBK004 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAB201 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RAB204 included  dna_r9.4.1_450bps_hac
FLO-MIN106 SQK-RPB004 included  dna_r9.4.1_450bps_hac
FLO-MIN106 VSK-VMK001 included  dna_r9.4.1_450bps_hac
FLO-MIN106 VSK-VMK002 included  dna_r9.4.1_450bps_hac
FLO-PRO001 SQK-RNA002           rna_r9.4.1_70bps_hac_prom
FLO-PRO002 SQK-RNA002           rna_r9.4.1_70bps_hac_prom
```

(continues on next page)

(continued from previous page)

```
FLO-FLG001 SQK-RNA001           rna_r9.4.1_70bps_hac
FLO-FLG001 SQK-RNA002           rna_r9.4.1_70bps_hac
FLO-MIN106 SQK-RNA001           rna_r9.4.1_70bps_hac
FLO-MIN106 SQK-RNA002           rna_r9.4.1_70bps_hac
FLO-MIN107 SQK-RNA001           rna_r9.4.1_70bps_hac
FLO-MIN107 SQK-RNA002           rna_r9.4.1_70bps_hac
FLO-MIN107 SQK-DCS108           dna_r9.5_450bps
FLO-MIN107 SQK-DCS109           dna_r9.5_450bps
FLO-MIN107 SQK-LRK001           dna_r9.5_450bps
FLO-MIN107 SQK-LSK108           dna_r9.5_450bps
FLO-MIN107 SQK-LSK109           dna_r9.5_450bps
FLO-MIN107 SQK-LSK308           dna_r9.5_450bps
FLO-MIN107 SQK-LSK309           dna_r9.5_450bps
FLO-MIN107 SQK-LSK319           dna_r9.5_450bps
FLO-MIN107 SQK-LWP001           dna_r9.5_450bps
FLO-MIN107 SQK-PCS108           dna_r9.5_450bps
FLO-MIN107 SQK-PCS109           dna_r9.5_450bps
FLO-MIN107 SQK-PSK004           dna_r9.5_450bps
FLO-MIN107 SQK-RAD002           dna_r9.5_450bps
FLO-MIN107 SQK-RAD003           dna_r9.5_450bps
FLO-MIN107 SQK-RAD004           dna_r9.5_450bps
FLO-MIN107 SQK-RAS201           dna_r9.5_450bps
FLO-MIN107 SQK-RLI001           dna_r9.5_450bps
FLO-MIN107 VSK-VBK001           dna_r9.5_450bps
FLO-MIN107 VSK-VSK001           dna_r9.5_450bps
FLO-MIN107 VSK-VSK002           dna_r9.5_450bps
FLO-MIN107 SQK-LWB001 included  dna_r9.5_450bps
FLO-MIN107 SQK-PBK004 included  dna_r9.5_450bps
FLO-MIN107 SQK-RAB201 included  dna_r9.5_450bps
FLO-MIN107 SQK-RAB204 included  dna_r9.5_450bps
FLO-MIN107 SQK-RBK001 included  dna_r9.5_450bps
FLO-MIN107 SQK-RBK004 included  dna_r9.5_450bps
FLO-MIN107 SQK-RLB001 included  dna_r9.5_450bps
FLO-MIN107 SQK-RPB004 included  dna_r9.5_450bps
FLO-MIN107 VSK-VMK001 included  dna_r9.5_450bps
FLO-MIN107 VSK-VMK002 included  dna_r9.5_450bps
FLO-PRO111 SQK-LSK109           dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-LSK109-XL          dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-LSK110           dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-DCS109           dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-PCS109           dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-PRC109           dna_r10.3_450bps_hac_prom
FLO-PRO111 SQK-PCB109 included  dna_r10.3_450bps_hac_prom
```

Our dataset was generated using the FLO-MIN106 flowcell, and the LSK109 kit, pick the appropriate model.

Note: You need to add ".cfg" to the model name in the command line or the basecaller exits with an error because the .cfg could not be found.

Try to get the basecalling running, use the following optional arguments:

```
--compress_fastq                Compress fastq output files with gzip.
--num_callers arg               Number of parallel basecallers to create.
--cpu_threads_per_caller arg    Number of CPU worker threads per basecaller.
```

Important: You have 14 CPUs in your virtual machine. The number of callers multiplied with the number of CPU threads per caller should not exceed 14.

The output directory should be named:

```
~/workdir/data_artic/basecall_tiny/
```

If you are stuck, you can skip to the next page and get help with the command.

References

guppy https://nanoporetech.com/

1.3.2 Basecalling with Guppy(2)

We need to specify the following options:

What?	parameter	Our value
The config file for our flowcell/kit combination	-c	dna_r9.4.1_450bps_hac_model.cfg
Compress the fastq output	–compress_fastq	
The full path to the directory where the raw read files are located	-i	~/workdir/data_artic/barcode_tiny/
The full path to the directory where the basecalled files will be saved	-s	~/workdir/data_artic/basecall_tiny/
How many worker threads you are using	–cpu_threads_per_caller	14
Number of parallel basecallers to create	–num_callers	1

Our complete command line using these parameters is:

```
guppy_basecaller_cpu --compress_fastq -i  ~/workdir/data_artic/barcode_tiny/ -s ~/
↪workdir/data_artic/basecall_tiny/ --cpu_threads_per_caller 14 --num_callers 1 -c␣
↪dna_r9.4.1_450bps_hac.cfg
```

References

guppy https://nanoporetech.com/

1.3.3 Inspect the output

The directory contains the following output:

```
ls -l ~/workdir/data_artic/basecall_tiny

total 4544
-rw-rw-r-- 1 ubuntu ubuntu 2356539 Nov  4 10:03 fastq_runid_
↪ca07d29b288c9f8881387334d8fc7d195fb11339_0_0.fastq.gz
-rw-rw-r-- 1 ubuntu ubuntu 1209378 Nov  4 10:03 guppy_basecaller_log-2020-11-04_09-54-
↪21.log
-rw-rw-r-- 1 ubuntu ubuntu  969796 Nov  4 10:03 sequencing_summary.txt
-rw-rw-r-- 1 ubuntu ubuntu  108243 Nov  4 10:03 sequencing_telemetry.js
```

So we have one fastq file in our directory - since we started with one fast5 file. Ususally, we should merge all resulting fastq files into a single file:

```
cat ~/workdir/data_artic/basecall_tiny/*.fastq.gz > ~/workdir/data_artic/basecall_
↪tiny.fastq.gz
```

In order to get the number of reads in our fastq file, we can count the number of lines and divide by 4:

```
zcat ~/workdir/data_artic/basecall_tiny.fastq.gz | wc -l | awk '{print $1/4}'
```

And to get the number of bases:

```
zcat ~/workdir/data_artic/basecall_tiny.fastq.gz | paste - - - - | cut -f 2 | tr -d
↪'\n' | wc -c
```

With a genome size of ~30k we get a coverage of:

```
zcat ~/workdir/data_artic/basecall_tiny.fastq.gz | paste - - - - | cut -f 2 | tr -d
↪'\n' | wc -c | awk '{print $1/30000}'
```

1.4 Data Quality Assessment

In the following, we will assess the data quality by looking at the sequencing effort, the raw reads and the data quality as reported by the sequencing instrument (using MinIONQC and FastQC) as well as inferring the actual data quality by aligning the reads to the reference genome (read mapping) and using Qualimap for error statistics.

1.4.1 MinIONQC

First of all, we will use MinIONQC to quality control our data.

MinIONQC is developed by Rob Lanfear:

```
R Lanfear, M Schalamun, D Kainer, W Wang, B Schwessinger; MinIONQC: fast and simple␣
↪quality control for MinION sequencing data, Bioinformatics, , bty654, https://doi.
↪org/10.1093/bioinformatics/bty654
```

Script collection that will generate a range of diagnostic plots for quality control of sequencing data from Oxford Nanopore's MinION sequencer.

MinIONQC works directly with the sequencing_summary.txt files produced by ONT's Albacore or Guppy base callers. This allows MinIONQC for quick-and-easy comparison of data from one or multiple flowcells.

Complete manual can be looked up at: https://github.com/roblanf/minion_qc

Usage:

```
MinIONQC.R --help
Usage: MinIONQC.R [options]
```

Options:

-h, --help Show this help message and exit

-i INPUT, --input=INPUT Input file or directory (required). Either a full path to a sequence_summary.txt file, or a full path to a directory containing one or more such files. In the latter case the directory is searched recursively.

-o OUTPUTDIRECTORY, --outputdirectory=OUTPUTDIRECTORY Output directory (optional, default is the same as the input directory). If a single sequencing_summary.txt file is passed as input, then the output directory will contain just the plots associated with that file. If a directory containing more than one sequencing_summary.txt files is passed as input, then the plots will be put into sub-directories that have the same names as the parent directories of each sequencing_summary.txt file

-q QSCORE_CUTOFF, --qscore_cutoff=QSCORE_CUTOFF The cutoff value for the mean Q score of a read (default 7). Used to create separate plots for reads above and below this threshold

-p PROCESSORS, --processors=PROCESSORS Number of processors to use for the anlaysis (default 1). Only helps when you are analysing more than one sequencing_summary.txt file at a time

-s SMALLFIGURES, --smallfigures=SMALLFIGURES TRUE or FALSE (the default). When true, MinIONQC will output smaller figures, e.g. suitable for publications or presentations. The default is to produce larger figures optimised for display on screen. Some figures just require small text, and cannot be effectively resized.

Run MinionQC to get some statistics for your data. The output directory should be:

```
~/workdir/data_artic/MinIONQC/
```

If you are stuck, go to the next page for help with the command.

1.4.2 MinIONQC(2)

We need to run Rscript with MinIONQC.R:

```
MinIONQC.R
```

. . . and specify the following options:

What?	parameter	Our value
The input directory	-i	~/workdir/data_artic/basecall_tiny/
The output directory	-o	~/workdir/data_artic/MinIONQC/
The number of processors to be used	-p	14

The complete command is:

```
cd ~/workdir/data_artic/
MinIONQC.R -i basecall_tiny -o MinIONQC -p 14
```

This will create several analysis plots. After that, you can load the plots in your web browser by using a file browser. You could use:

```
firefox ~/workdir/data_artic/MinIONQC/basecall_tiny/*.png
```

to load all images at once in firefox.

Again, check out the corresponding home page to learn more about all generated results: https://github.com/roblanf/minion_qc

1.4.3 FastQC

Get Illumina data

We have prepared some Illumina data for comparison with Nanopore data for you. Get the data with:

```
cd ~/workdir
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/Illumina.
↪tar.gz
```

Then unpack and remove the tar file:

```
cd ~/workdir/
tar -xzvf Illumina.tar.gz
rm Illumina.tar.gz
```

Check, what's in the Illumina directory:

```
ls -l ~/workdir/Illumina/
total 109108
-rw-r--r-- 1 ubuntu ubuntu 53544266 Nov  4 14:04 D0003_S3_L001_R1_001.fastq.gz
-rw-r--r-- 1 ubuntu ubuntu 58178627 Nov  4 14:04 D0003_S3_L001_R2_001.fastq.gz
```

FastQC

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 main functions of FastQC are

- Import of data from BAM, SAM or FastQ files (any variant)
- Providing a quick overview to tell you in which areas there may be problems
- Summary graphs and tables to quickly assess your data
- Export of results to an HTML based permanent report
- Offline operation to allow automated generation of reports without running the interactive application

You can run FastQC interactively or using ht CLI, which offers the following options:

```
fastqc --help

SYNOPSIS

  fastqc seqfile1 seqfile2 .. seqfileN

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

OPTIONS:

  -o --outdir     Create all output files in the specified output directory.
                  Please note that this directory must exist as the program
                  will not create it.  If this option is not set then the
                  output file for each sequence file is created in the same
```

(continues on next page)

(continued from previous page)

```
                directory as the sequence file which was processed.

 --casava       Files come from raw casava output. Files in the same sample
                group (differing only by the group number) will be analysed
                as a set rather than individually. Sequences with the filter
                flag set in the header will be excluded from the analysis.
                Files must have the same names given to them by casava
                (including being gzipped and ending with .gz) otherwise they
                won't be grouped together correctly.

 --nano         Files come from naopore sequences and are in fast5 format. In
                this mode you can pass in directories to process and the program
                will take in all fast5 files within those directories and produce
                a single output file from the sequences found in all files.

 --nofilter     If running with --casava then don't remove read flagged by
                casava as poor quality when performing the QC analysis.

 --nogroup      Disable grouping of bases for reads >50bp. All reports will
                show data for every base in the read.  WARNING: Using this
                option will cause fastqc to crash and burn if you use it on
                really long reads, and your plots may end up a ridiculous size.
                You have been warned!

 -f --format    Bypasses the normal sequence file format detection and
                forces the program to use the specified format.  Valid
                formats are bam,sam,bam_mapped,sam_mapped and fastq

 -t --threads   Specifies the number of files which can be processed
                simultaneously.  Each thread will be allocated 250MB of
                memory so you shouldn't run more threads than your
                available memory will cope with, and not more than
                6 threads on a 32 bit machine

 -c             Specifies a non-default file which contains the list of
 --contaminants contaminants to screen overrepresented sequences against.
                The file must contain sets of named contaminants in the
                form name[tab]sequence.  Lines prefixed with a hash will
                be ignored.

 -a             Specifies a non-default file which contains the list of
 --adapters     adapter sequences which will be explicity searched against
                the library. The file must contain sets of named adapters
                in the form name[tab]sequence.  Lines prefixed with a hash
                will be ignored.

 -l             Specifies a non-default file which contains a set of criteria
 --limits       which will be used to determine the warn/error limits for the
                various modules.  This file can also be used to selectively
                remove some modules from the output all together.  The format
                needs to mirror the default limits.txt file found in the
                Configuration folder.

-k --kmers      Specifies the length of Kmer to look for in the Kmer content
                module. Specified Kmer length must be between 2 and 10. Default
                length is 7 if not specified.
```

(continues on next page)

(continued from previous page)

```
-q --quiet       Supress all progress messages on stdout and only report errors.

-d --dir         Selects a directory to be used for temporary files written when
                 generating report images. Defaults to system temp directory if
                 not specified.
```

See the FastQC home page for more info.

QA with FastQC

Your task is to run FastQC on the Illumina data and on your basecalled samples. Create a new directory for the results:

```
mkdir ~/workdir/fastqc
```

The results should be stored in:

```
~/workdir/fastqc/illumina/
```

and:

```
~/workdir/fastqc/basecall_tiny
```

You need to create the result directories before runnnig FastQC, because FastQC will note create them.

If you are stuck, go to the next page to get further help.

References

FastQC https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

1.4.4 FastQC(2)

We need to run fastqc with the following command:

```
fastqc
```

and with the following options:

What?	parameter	Our value
The input directory	positional	~/workdir/data_artic/basecall_tiny/ or ~/workdir/Illumina/*.fastq.gz
The output directory	-o	~/workdir/fastqc/basecall_tiny/ or ~/workdir/fastqc/illumina/
The number of threads to be used	-t	14

Go to your workdir fist:

```
cd ~/workdir
```

Then create folders for the results (fastqc will not create them):

```
mkdir -p ~/workdir/fastqc/basecall_tiny
mkdir -p ~/workdir/fastqc/illumina
```

The run fastqc for Illumina data and on Nanopore data:

```
fastqc -t 14 -o ~/workdir/fastqc/illumina ~/workdir/Illumina/*.fastq.gz

fastqc -t 14 -o ~/workdir/fastqc/basecall_tiny ~/workdir/data_artic/basecall_tiny.
↪fastq.gz
```

After that, you can load the reports in your web browser. Open a file browser, go to your workdir/fastqc/ directory and double click the html file. Or use the command line:

```
firefox ~/workdir/fastqc/illumina*.html

firefox ~/workdir/fastqc/basecall_tiny/*.html
```

Look out for the differences between Illumina data und Nanopore data. Is there Addapter contamination in your read data?

You should also check out the FastQC home page for examples of reports including bad data.

Note: When you run fastqc without "-o", the results of fastqc are located in the directory that contains the reads.

So the first bases may indicate an adapter contamination. For workflows including *de novo* assembly refined with nanopolish or medaka adapter trimming is not necessary, but in other workflow scenarios this can be important to do and good there are tools which can handle this, as e.g. **porechop**.

References

FastQC https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

1.4.5 Porechop

As we see some strange GC content at the 5' end of our nanopore reads, the first bases may indicate an adaptor contamination. For workflows including de novo assembly refined with nanopolish or medaka adaptor trimming is not necessary, but in other workflow scenarios (like mapping for the ARTIC pipeline) this can be important to do and good there are tools which can handle this, as e.g. **porechop**.

Porechop is a tool for finding and removing adapters from Oxford Nanopore reads. Adapters on the ends of reads are trimmed off, and when a read has an adapter in its middle, it is treated as chimeric and chopped into separate reads. Porechop performs thorough alignments to effectively find adapters, even at low sequence identity:

```
porechop --help

usage: porechop -i INPUT [-o OUTPUT] [--format {auto,fasta,fastq,fasta.gz,fastq.gz}]␣
↪[-v VERBOSITY] [-t THREADS] [-b BARCODE_DIR] [--barcode_threshold BARCODE_
↪THRESHOLD] [--barcode_diff BARCODE_DIFF] [--require_two_barcodes] [--untrimmed] [--
↪discard_unassigned]
                 [--adapter_threshold ADAPTER_THRESHOLD] [--check_reads CHECK_READS] [-
↪-scoring_scheme SCORING_SCHEME] [--end_size END_SIZE] [--min_trim_size MIN_TRIM_
↪SIZE] [--extra_end_trim EXTRA_END_TRIM] [--end_threshold END_THRESHOLD] [--no_
↪split] [--discard_middle]
                 [--middle_threshold MIDDLE_THRESHOLD] [--extra_middle_trim_good_side␣
↪EXTRA_MIDDLE_TRIM_GOOD_SIDE] [--extra_middle_trim_bad_side EXTRA_MIDDLE_TRIM_BAD_
↪SIDE] [--min_split_read_size MIN_SPLIT_READ_SIZE] [-h] [--version]
```

(continues on next page)

(continued from previous page)

```
Porechop: a tool for finding adapters in Oxford Nanopore reads, trimming them from␣
↪the ends and splitting reads with internal adapters

Main options:
  -i INPUT, --input INPUT               FASTA/FASTQ of input reads or a directory␣
↪which will be recursively searched for FASTQ files (required)
  -o OUTPUT, --output OUTPUT            Filename for FASTA or FASTQ of trimmed reads␣
↪(if not set, trimmed reads will be printed to stdout)
  --format {auto,fasta,fastq,fasta.gz,fastq.gz}
                                        Output format for the reads - if auto, the␣
↪format will be chosen based on the output filename or the input read format␣
↪(default: auto)
  -v VERBOSITY, --verbosity VERBOSITY   Level of progress information: 0 = none, 1 =␣
↪some, 2 = lots, 3 = full - output will go to stdout if reads are saved to a file␣
↪and stderr if reads are printed to stdout (default: 1)
  -t THREADS, --threads THREADS         Number of threads to use for adapter␣
↪alignment (default: 16)

Barcode binning settings:
  Control the binning of reads based on barcodes (i.e. barcode demultiplexing)

  -b BARCODE_DIR, --barcode_dir BARCODE_DIR
                                        Reads will be binned based on their barcode␣
↪and saved to separate files in this directory (incompatible with --output)
  --barcode_threshold BARCODE_THRESHOLD
                                        A read must have at least this percent␣
↪identity to a barcode to be binned (default: 75.0)
  --barcode_diff BARCODE_DIFF           If the difference between a read's best␣
↪barcode identity and its second-best barcode identity is less than this value, it␣
↪will not be put in a barcode bin (to exclude cases which are too close to call)␣
↪(default: 5.0)
  --require_two_barcodes                Reads will only be put in barcode bins if␣
↪they have a strong match for the barcode on both their start and end (default: a␣
↪read can be binned with a match at its start or end)
  --untrimmed                           Bin reads but do not trim them (default: trim␣
↪the reads)
  --discard_unassigned                  Discard unassigned reads (instead of creating␣
↪a "none" bin) (default: False)

Adapter search settings:
  Control how the program determines which adapter sets are present

  --adapter_threshold ADAPTER_THRESHOLD
                                        An adapter set has to have at least this␣
↪percent identity to be labelled as present and trimmed off (0 to 100) (default: 90.
↪0)
  --check_reads CHECK_READS             This many reads will be aligned to all␣
↪possible adapters to determine which adapter sets are present (default: 10000)
  --scoring_scheme SCORING_SCHEME       Comma-delimited string of alignment scores:␣
↪match, mismatch, gap open, gap extend (default: 3,-6,-5,-2)

End adapter settings:
  Control the trimming of adapters from read ends

  --end_size END_SIZE                   The number of base pairs at each end of the␣
↪read which will be searched for adapter sequences (default: 150)
```

(continues on next page)

(continued from previous page)

```
  --min_trim_size MIN_TRIM_SIZE         Adapter alignments smaller than this will be␣
↪ignored (default: 4)
  --extra_end_trim EXTRA_END_TRIM       This many additional bases will be removed␣
↪next to adapters found at the ends of reads (default: 2)
  --end_threshold END_THRESHOLD         Adapters at the ends of reads must have at␣
↪least this percent identity to be removed (0 to 100) (default: 75.0)

Middle adapter settings:
  Control the splitting of read from middle adapters

  --no_split                            Skip splitting reads based on middle adapters␣
↪(default: split reads when an adapter is found in the middle)
  --discard_middle                      Reads with middle adapters will be discarded␣
↪(default: reads with middle adapters are split) (required for reads to be used with␣
↪Nanopolish, this option is on by default when outputting reads into barcode bins)
  --middle_threshold MIDDLE_THRESHOLD   Adapters in the middle of reads must have at␣
↪least this percent identity to be found (0 to 100) (default: 90.0)
  --extra_middle_trim_good_side EXTRA_MIDDLE_TRIM_GOOD_SIDE
                                        This many additional bases will be removed␣
↪next to middle adapters on their "good" side (default: 10)
  --extra_middle_trim_bad_side EXTRA_MIDDLE_TRIM_BAD_SIDE
                                        This many additional bases will be removed␣
↪next to middle adapters on their "bad" side (default: 100)
  --min_split_read_size MIN_SPLIT_READ_SIZE
                                        Post-split read pieces smaller than this many␣
↪base pairs will not be outputted (default: 1000)

Help:
  -h, --help                            Show this help message and exit
  --version                             Show program's version number and exit
```

Use **porechop** on the basecalled dataset (basecall_tiny.fastq.gz). The results should be stored as:

```
~/workdir/data_artic/basecall_tiny_porechopped.fastq.gz
```

Also use the following options:

```
-t 14 (to use 14 threads)
-v 2 (to get more verbose output)
```

In addition, you should write the output of porechop into a file called "porechop.log" for later inspection.

When you are done running porechop, run **fastqc** again on the trimmed data for comparison, the results of this fastqc run should be stored in:

```
~/workdir/fastqc/basecall_tiny_porechopped/
```

Again - if you are stuck, check the next page for futher help/instructions.

References

Porechop https://github.com/rrwick/Porechop

FastQC https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

1.4.6 Porechop(2)

We need to run the following command:

```
porechop
```

with the following parameters

Go to your data_artic directory first:

```
cd ~/workdir/data_artic/
```

Then run porechop:

```
porechop -i ~/workdir/data_artic/basecall_tiny.fastq.gz -t 14 -v 2 -o ~/workdir/data_
↪artic/basecall_tiny_porechopped.fastq.gz |tee ~/workdir/data_artic/porechop.log
```

Task: Inspect the log file -have adapters been removed?

And finally, run FastQC again, to compare with the untrimmed data:

```
mkdir ~/workdir/fastqc/basecall_tiny_trimmed/
fastqc -t 14 -o  ~/workdir/fastqc/basecall_tiny_trimmed ~/workdir/data_artic/basecall_
↪tiny_porechopped.fastq.gz
```

References

FastQC https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Porechop https://github.com/rrwick/Porechop

1.4.7 Read mapping to a reference

We will now map the Nanopore and Illumina reads to the Wuhan reference, have a look on the reads and read distribution, and generate error profiles from the mappings. Due to mutations, this is, of course not 100% accurate, but gives an impression.

Get the reference

First, download the reference from the object store:

```
cd ~/workdir/
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/wuhan.
↪fasta.gz
```

And unzip:

```
gunzip wuhan.fasta.gz
```

Mapping the data

We will use Minimap2 to map the reads to the reference.

Minimap2 is a versatile sequence alignment program that aligns DNA or mRNA sequences against a large reference database. Typical use cases include: (1) mapping PacBio or Oxford Nanopore genomic reads to the human genome; (2) finding overlaps between long reads with error rate up to ~15%; (3) splice-aware alignment of PacBio Iso-Seq or Nanopore cDNA or Direct RNA reads against a reference genome; (4) aligning Illumina single- or paired-end reads; (5) assembly-to-assembly alignment; (6) full-genome alignment between two closely related species with divergence below ~15%.

Get help for minimap2:

```
minimap2 --help

Usage: minimap2 [options] <target.fa>|<target.idx> [query.fa] [...]
Options:
  Indexing:
    -H           use homopolymer-compressed k-mer (preferrable for PacBio)
    -k INT       k-mer size (no larger than 28) [15]
    -w INT       minimizer window size [10]
    -I NUM       split index for every ~NUM input bases [4G]
    -d FILE      dump index to FILE []
  Mapping:
    -f FLOAT     filter out top FLOAT fraction of repetitive minimizers [0.0002]
    -g NUM       stop chain enlongation if there are no minimizers in INT-bp [5000]
    -G NUM       max intron length (effective with -xsplice; changing -r) [200k]
    -F NUM       max fragment length (effective with -xsr or in the fragment mode)␣
↪[800]
    -r NUM       bandwidth used in chaining and DP-based alignment [500]
    -n INT       minimal number of minimizers on a chain [3]
    -m INT       minimal chaining score (matching bases minus log gap penalty) [40]
    -X           skip self and dual mappings (for the all-vs-all mode)
    -p FLOAT     min secondary-to-primary score ratio [0.8]
    -N INT       retain at most INT secondary alignments [5]
  Alignment:
    -A INT       matching score [2]
    -B INT       mismatch penalty [4]
    -O INT[,INT] gap open penalty [4,24]
    -E INT[,INT] gap extension penalty; a k-long gap costs min{O1+k*E1,O2+k*E2} [2,1]
    -z INT[,INT] Z-drop score and inversion Z-drop score [400,200]
    -s INT       minimal peak DP alignment score [80]
    -u CHAR      how to find GT-AG. f:transcript strand, b:both strands, n:don't␣
↪match GT-AG [n]
  Input/Output:
    -a           output in the SAM format (PAF by default)
    -o FILE      output alignments to FILE [stdout]
    -L           write CIGAR with >65535 ops at the CG tag
    -R STR       SAM read group line in a format like '@RG\tID:foo\tSM:bar' []
    -c           output CIGAR in PAF
    --cs[=STR]   output the cs tag; STR is 'short' (if absent) or 'long' [none]
    --MD         output the MD tag
    --eqx        write =/X CIGAR operators
    -Y           use soft clipping for supplementary alignments
    -t INT       number of threads [3]
    -K NUM       minibatch size for mapping [500M]
    --version    show version number
  Preset:
    -x STR       preset (always applied before other options; see minimap2.1 for␣
↪details) []
                 - map-pb/map-ont: PacBio/Nanopore vs reference mapping
                 - ava-pb/ava-ont: PacBio/Nanopore read overlap
```

(continues on next page)

(continued from previous page)

```
                - asm5/asm10/asm20: asm-to-ref mapping, for ~0.1/1/5% sequence␣
↪divergence
                - splice: long-read spliced alignment
                - sr: genomic short-read mapping
```

Create a directory for the mapping results in:

```
~/workdir/mappings/
```

The result of your mappings should be named:

```
~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sam
```

Use the following additional options:

```
-x <appropriate preset>
-t <number of threads>
-a (to get sam output)
```

When you are done (or stuck) try to map the Illumina data before you proceed to the next page.

For the Illumina data, we use bwa, as this one is more suited for this kind of data. But before we can do so, we need to create an index structure on the reference:

```
Usage:   bwa index [options] <in.fasta>

Options: -a STR    BWT construction algorithm: bwtsw, is or rb2 [auto]
         -p STR    prefix of the index [same as fasta name]
         -b INT    block size for the bwtsw algorithm (effective with -a bwtsw)␣
↪[10000000]
         -6        index files named as <in.fasta>.64.* instead of <in.fasta>.*

Warning: `-a bwtsw' does not work for short genomes, while `-a is' and
         `-a div' do not work not for long genomes.
```

Then do the mapping:

```
Usage: bwa mem [options] <idxbase> <in1.fq> [in2.fq]

Algorithm options:

       -t INT        number of threads [1]
       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT␣
↪[1.5]
       -y INT        seed occurrence for the 3rd round seeding [20]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest␣
↪overlapping chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use

Scoring options:
```

(continues on next page)

(continued from previous page)

```
       -A INT        score for a sequence match, which scales options -TdBOELU unless␣
↪overridden [1]
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

       -x STR        read type. Setting -x changes multiple parameters unless␣
↪overridden [null]
                     pacbio: -k17 -W40 -r10 -A1 -B1 -O1 -E1 -L0  (PacBio reads to ref)
                     ont2d: -k14 -W20 -r10 -A1 -B1 -O1 -E1 -L0  (Oxford Nanopore 2D-
↪reads to ref)
                     intractg: -B9 -O16 -L5  (intra-species contigs to ref)

Input/output options:

       -p            smart pairing (ignoring in2.fq)
       -R STR        read group header line such as '@RG\tID:foo\tSM:bar' [null]
       -H STR/FILE   insert STR to header if it starts with @; or insert lines in␣
↪FILE [null]
       -o FILE       sam file to output results to [stdout]
       -j            treat ALT contigs as part of the primary assembly (i.e. ignore
↪<idxbase>.alt file)
       -5            for split alignment, take the alignment with the smallest␣
↪coordinate as primary
       -q            don't modify mapQ of supplementary alignments
       -K INT        process INT input bases in each batch regardless of nThreads␣
↪(for reproducibility) []

       -v INT        verbosity level: 1=error, 2=warning, 3=message, 4+=debugging [3]
       -T INT        minimum score to output [30]
       -h INT[,INT]  if there are <INT hits with score >80% of the max score, output␣
↪all in XA [5,200]
       -a            output all alignments for SE or unpaired PE
       -C            append FASTA/FASTQ comment to SAM output
       -V            output the reference FASTA header in the XR tag
       -Y            use soft clipping for supplementary alignments
       -M            mark shorter split hits as secondary

       -I FLOAT[,FLOAT[,INT[,INT]]]
                     specify the mean, standard deviation (10% of the mean if absent),
↪ max
                     (4 sigma from the mean if absent) and min of the insert size␣
↪distribution.
                     FR orientation only. [inferred]
```

Use:

```
-t 14
```

to use 14 threads. No other options, redirect output into a file called:

```
~/workdir/mappings/illumina_vs_wuhan.sam
```

Again, if you are stuck, get help on the next page.

References

Minimap2 https://github.com/lh3/minimap2

BWA http://bio-bwa.sourceforge.net/

1.4.8 Read mapping to a reference (2)

Mapping the Nanopore data with Minimap2

We use:

```
minimap2
```

to compute the mapping and add the following parameters:

What?	parameter	Our value
The reference file	positional (1)	~/workdir/wuhan.fasta
The input file	positional (2)	~/workdir/data_artic/basecall_tiny_porechopped.fastq.gz
The output file	-o	~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sam
The number of threads to be used	-t	14
The preset for Nanopore reads	-x	map-ont
Get sam output	-a	

Create the mapping folder first:

```
mkdir ~/workdir/mappings/
```

The complete commandline for minimap2 is:

```
minimap2 -t 14 -x map-ont -a -o ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.
↪sam ~/workdir/wuhan.fasta ~/workdir/data_artic/basecall_tiny_porechopped.fastq.gz
```

Mapping the Illumina data with bwa

First, we use:

```
bwa index ~/workdir/wuhan.fasta
```

to create an index on the reference.

Then, we run the actual mapping with:

```
bwa mem
```

and the following parameters:

What?	parameter	Our value
The reference file	positional (1)	~/workdir/wuhan.fasta
The input file(s)	positional (2)	~/workdir/Illumina/D0003_S3_L001_R1_001.fastq.gz and ~/workdir/illumina/D0003_S3_L001_R2_001.fastq.gz
The output file	redirect with >	~/workdir/mappings/illumina_vs_wuhan.sam
The number of threads to be used	-t	14

The complete commandline for bwa mem is:

```
bwa mem -t 14  ~/workdir/wuhan.fasta  ~/workdir/Illumina/D0003_S3_L001_R1_001.fastq.
↪gz ~/workdir/Illumina/D0003_S3_L001_R2_001.fastq.gz > ~/workdir/mappings/illumina_
↪vs_wuhan.sam
```

Converting sam files to sorted and indexed bam files

In order to read the mapping files into a genome browser, we need to convert the .sam files to sorted .bam files. This is done, using 3 different tools from samtools, the first converts sam to bam:

```
Usage: samtools view [options] <in.bam>|<in.sam>|<in.cram> [region ...]

Options:
  -b       output BAM
  -C       output CRAM (requires -T)
  -1       use fast BAM compression (implies -b)
  -u       uncompressed BAM output (implies -b)
  -h       include header in SAM output
  -H       print SAM header only (no alignments)
  -c       print only the count of matching records
  -o FILE  output file name [stdout]
  -U FILE  output reads not selected by filters to FILE [null]
  -t FILE  FILE listing reference names and lengths (see long help) [null]
  -X       include customized index file
  -L FILE  only include reads overlapping this BED FILE [null]
  -r STR   only include reads in read group STR [null]
  -R FILE  only include reads with read group listed in FILE [null]
  -d STR:STR
           only include reads with tag STR and associated value STR [null]
  -D STR:FILE
           only include reads with tag STR and associated values listed in
           FILE [null]
  -q INT   only include reads with mapping quality >= INT [0]
  -l STR   only include reads in library STR [null]
  -m INT   only include reads with number of CIGAR operations consuming
           query sequence >= INT [0]
  -f INT   only include reads with all  of the FLAGs in INT present [0]
  -F INT   only include reads with none of the FLAGS in INT present [0]
  -G INT   only EXCLUDE reads with all  of the FLAGs in INT present [0]
  -s FLOAT subsample reads (given INT.FRAC option value, 0.FRAC is the
```

(continues on next page)

(continued from previous page)

```
          fraction of templates/read pairs to keep; INT part sets seed)
  -M       use the multi-region iterator (increases the speed, removes
          duplicates and outputs the reads as they are ordered in the file)
  -x STR   read tag to strip (repeatable) [null]
  -B       collapse the backward CIGAR operation
  -?       print long help, including note about region specification
  -S       ignored (input format is auto-detected)
  --no-PG  do not add a PG line
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
  -O, --output-fmt FORMAT[,OPT[=VAL]]...
               Specify output format (SAM, BAM, CRAM)
      --output-fmt-option OPT[=VAL]
               Specify a single output file format option in the form
               of OPTION or OPTION=VALUE
  -T, --reference FILE
               Reference sequence FASTA FILE [null]
  -@, --threads INT
               Number of additional threads to use [0]
      --write-index
               Automatically index the output files [off]
      --verbosity INT
               Set level of verbosity
```

For our purpose, we need the options:

```
-b for sam to bam conversion
```

Redirect the output into files with name (or redirect directly to samtools sort - see further below):

```
~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.bam
or
~/workdir/mappings/illumina_vs_wuhan.bam
```

Then sort the bam file with samtools sort:

```
Usage: samtools sort [options...] [in.bam]
Options:
  -l INT     Set compression level, from 0 (uncompressed) to 9 (best)
  -u         Output uncompressed data (equivalent to -l 0)
  -m INT     Set maximum memory per thread; suffix K/M/G recognized [768M]
  -M         Use minimiser for clustering unaligned/unplaced reads
  -K INT     Kmer size to use for minimiser [20]
  -n         Sort by read name (not compatible with samtools index command)
  -t TAG     Sort by value of TAG. Uses position as secondary index (or read name if -
↪n is set)
  -o FILE    Write final output to FILE rather than standard output
  -T PREFIX  Write temporary files to PREFIX.nnnn.bam
  --no-PG    do not add a PG line
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
  -O, --output-fmt FORMAT[,OPT[=VAL]]...
               Specify output format (SAM, BAM, CRAM)
      --output-fmt-option OPT[=VAL]
               Specify a single output file format option in the form
```

(continues on next page)

(continued from previous page)

```
                of OPTION or OPTION=VALUE
      --reference FILE
                Reference sequence FASTA FILE [null]
  -@, --threads INT
                Number of additional threads to use [0]
      --verbosity INT
                Set level of verbosity
```

Your sorted bam files should have the name:

```
~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sorted.bam
or
~/workdir/mappings/illumina_vs_wuhan.sorted.bam
```

Then index the sorted bam file:

```
Usage: samtools index [-bc] [-m INT] <in.bam> [out.index]
Options:
  -b       Generate BAI-format index for BAM files [default]
  -c       Generate CSI-format index for BAM files
  -m INT   Set minimum interval size for CSI indices to 2^INT [14]
  -@ INT   Sets the number of threads [none]
```

If you are stuck - get help on the next page.

References

Minimap2 https://github.com/lh3/minimap2

BWA http://bio-bwa.sourceforge.net/

samtools http://www.htslib.org

1.4.9 Read mapping to a reference (3)

SAM to BAM conversion with samtools

To create a sorted BAM file from a SAM file we can use:

```
samtools view -b ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sam | samtools␣
↪sort - > ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sorted.bam
```

And for the Illumina data:

```
samtools view -b ~/workdir/mappings/illumina_vs_wuhan.sam | samtools sort - > ~/
↪workdir/mappings/illumina_vs_wuhan.sorted.bam
```

Then create indizes on the sorted BAM files:

```
samtools index ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sorted.bam
and
samtools index ~/workdir/mappings/illumina_vs_wuhan.sorted.bam
```

Inspect the results using GenomeView

Then start the genome browser GenomeView:

```
java -jar ~/genomeview-N42.jar
```

Load the Wuhan-Reference and the mappings (the sorted BAM files) and inspect the mapping results. Have a look on the mapping quality in particular (Zoom in to see mismatches in alignment). You can load data with: File->Load data... then choose the appropriate files.

To see not only a fraction of the mappings go to File->Configuration->Short reads and change the "Maxiumum display depth of stacked reads" to a larger value.

References

samtools http://www.htslib.org

GenomeView https://genomeview.org/

1.4.10 Generating Error Profiles

Inferring error profiles using samtools

After mapping the reads on the reference Genome, we can infer various statistics as e.g., number of succesful aligned reads and bases, or number of mismatches and indels, and so on. For this you could easily use the tool collection **samtools**, which offers a range of simple CLI modules all operating on mapping output (SAM and BAM format). We will use the `stats` module now:

```
Usage: samtools stats [OPTIONS] file.bam
       samtools stats [OPTIONS] file.bam chr:from-to
Options:
    -c, --coverage <int>,<int>,<int>    Coverage distribution min,max,step [1,1000,1]
    -d, --remove-dups                   Exclude from statistics reads marked as␣
↪duplicates
    -X, --customized-index-file         Use a customized index file
    -f, --required-flag  <str|int>      Required flag, 0 for unset. See also␣
↪`samtools flags` [0]
    -F, --filtering-flag <str|int>      Filtering flag, 0 for unset. See also␣
↪`samtools flags` [0]
        --GC-depth <float>              the size of GC-depth bins (decreasing bin␣
↪size increases memory requirement) [2e4]
    -h, --help                          This help message
    -i, --insert-size <int>             Maximum insert size [8000]
    -I, --id <string>                   Include only listed read group or sample name
    -l, --read-length <int>             Include in the statistics only reads with the␣
↪given read length [-1]
    -m, --most-inserts <float>          Report only the main part of inserts [0.99]
    -P, --split-prefix <str>            Path or string prefix for filepaths output by␣
↪-S (default is input filename)
    -q, --trim-quality <int>            The BWA trimming parameter [0]
    -r, --ref-seq <file>                Reference sequence (required for GC-depth and␣
↪mismatches-per-cycle calculation).
    -s, --sam                           Ignored (input format is auto-detected).
    -S, --split <tag>                   Also write statistics to separate files split␣
↪by tagged field.
```

(continues on next page)

(continued from previous page)

```
    -t, --target-regions <file>         Do stats in these regions only. Tab-delimited␣
↪file chr,from,to, 1-based, inclusive.
    -x, --sparse                        Suppress outputting IS rows where there are␣
↪no insertions.
    -p, --remove-overlaps               Remove overlaps of paired-end reads from␣
↪coverage and base count computations.
    -g, --cov-threshold <int>           Only bases with coverage above this value␣
↪will be included in the target percentage computation [0]
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
      --reference FILE
               Reference sequence FASTA FILE [null]
  -@, --threads INT
               Number of additional threads to use [0]
      --verbosity INT
               Set level of verbosity
```

You can use:

```
-@ 14
```

to use more than one thread, but this shouldn't be necessary for the size of our dataset.

Forward the output of samtools stats into a file called:

```
~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.stats
```

Inspect the results using less or more. If you are stuck - get help on the next page.

References

Samtools http://samtools.sourceforge.net/

1.4.11 Generating Error Profiles(2)

Inferring error profiles using samtools

The command to run samtools stats is quite simple:

```
samtools stats ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sorted.bam >  ~/
↪workdir/mappings/basecall_tiny_porechopped_vs_wuhan.stats
```

and for Illumina:

```
samtools stats -@ 14 ~/workdir/mappings/illumina_vs_wuhan.sorted.bam >  ~/workdir/
↪mappings/illumina_vs_wuhan.stats
```

Inspect the files with:

```
less ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.stats
less ~/workdir/mappings/illumina_vs_wuhan.stats
```

Enhanced mapping statistics

To get a more in depth info on the actual accuracy of the data at hand, including the genome coverage, we're going to use a more comprehensive and interactive software comparable to FastQC which is called **Qualimap**. Qualimap has a tool "bamqc" for statistics on BAM files:

```
usage: qualimap bamqc -bam <arg> [-c] [-gd <arg>] [-gff <arg>] [-hm <arg>] [-ip]
       [-nr <arg>] [-nt <arg>] [-nw <arg>] [-oc <arg>] [-os] [-outdir <arg>]
       [-outfile <arg>] [-outformat <arg>] [-p <arg>] [-sd] [-sdmode <arg>]
 -bam <arg>                              Input mapping file in BAM format
 -c,--paint-chromosome-limits            Paint chromosome limits inside charts
 -gd,--genome-gc-distr <arg>             Species to compare with genome GC
                                         distribution. Possible values: HUMAN -
                                         hg19; MOUSE - mm9(default), mm10
 -gff,--feature-file <arg>               Feature file with regions of interest in
                                         GFF/GTF or BED format
 -hm <arg>                               Minimum size for a homopolymer to be
                                         considered in indel analysis (default is
                                         3)
 -ip,--collect-overlap-pairs             Activate this option to collect statistics
                                         of overlapping paired-end reads
 -nr <arg>                               Number of reads analyzed in a chunk
                                         (default is 1000)
 -nt <arg>                               Number of threads (default is 28)
 -nw <arg>                               Number of windows (default is 400)
 -oc,--output-genome-coverage <arg>      File to save per base non-zero coverage.
                                         Warning: large files are expected for
                                         large genomes
 -os,--outside-stats                     Report information for the regions outside
                                         those defined by feature-file  (ignored
                                         when -gff option is not set)
 -outdir <arg>                           Output folder for HTML report and raw
                                         data.
 -outfile <arg>                          Output file for PDF report (default value
                                         is report.pdf).
 -outformat <arg>                        Format of the output report (PDF, HTML or
                                         both PDF:HTML, default is HTML).
 -p,--sequencing-protocol <arg>          Sequencing library protocol:
                                         strand-specific-forward,
                                         strand-specific-reverse or
                                         non-strand-specific (default)
 -sd,--skip-duplicated                   Activate this option to skip duplicated
                                         alignments from the analysis. If the
                                         duplicates are not flagged in the BAM
                                         file, then they will be detected by
                                         Qualimap and can be selected for skipping.
 -sdmode,--skip-dup-mode <arg>           Specific type of duplicated alignments to
                                         skip (if this option is activated).
                                         0 : only flagged duplicates (default)
                                         1 : only estimated by Qualimap
                                         2 : both flagged and estimated



Special arguments:

    --java-mem-size  Use this argument to set Java memory heap size. Example:
                     qualimap bamqc -bam very_large_alignment.bam --java-mem-size=4G
```

Run:

```
qualimap bamqc
```

on the mapping files now (*.sorted.bam - Illumina and Nanopore mappings). Use the following options:

```
-nt 14
-nw 50000
-bam <input file>
-outdir <output directory>
```

The output folders should be named:

```
~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan_qualimap/
and
~/workdir/mappings/illumina_vs_wuhan_qualimap/
```

Help is available on the next page.

References

Samtools http://samtools.sourceforge.net/

QualiMap http://qualimap.bioinfo.cipf.es/doc_html/index.html

1.4.12 Generating Error Profiles(3)

Enhanced mapping statistics

We run:

```
qualimap bamqc
```

with the following parameters:

The complete commandline is:

```
qualimap bamqc -bam ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan.sorted.bam -
↪nw 5000 -nt 14 -c -outdir ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan_
↪qualimap/
```

and for Illumina:

```
qualimap bamqc -bam ~/workdir/mappings/illumina_vs_wuhan.sorted.bam -nw 5000 -nt 14 -
↪c -outdir ~/workdir/mappings/illumina_vs_wuhan_qualimap/
```

You can view the results with firefox:

```
firefox ~/workdir/mappings/basecall_tiny_porechopped_vs_wuhan_qualimap/qualimapReport.
↪html
or
firefox ~/workdir/mappings/illumina_vs_wuhan_qualimap/qualimapReport.html
```

References

QualiMap http://qualimap.bioinfo.cipf.es/doc_html/index.html

1.5 Recap Day 1

If you got lost on day 1, no problem, we will continue with larger read files, that contain approximately 10x the reads you used yesterday, which should be around 600x coverage. Make sure, that your data directory:

```
~/workdir/data_artic/
```

exists. If not, create it:

```
mkdir ~/workdir/data_artic/
```

All you need to go on is the basecalled and porechopped data (choose 1). You can get the data with:

```
cd ~/workdir/data_artic/
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪small_porechopped_01.fastq.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪small_porechopped_02.fastq.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪small_porechopped_03.fastq.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪small_porechopped_04.fastq.gz
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪small_porechopped_05.fastq.gz
```

Then rename your choice with:

```
mv ~/workdir/data_artic/basecall_small_porechopped_<number>.fastq.gz ~/workdir/data_
↪artic/basecall_small_porechopped.fastq.gz
```

When you are done, we can continue with the tutorial for day 2.

1.6 Assembly

We are going to create an assembly with canu and evaluate it with a mapping to the reference genome and with quast.

1.6.1 Assembly with canu

We will now try to assembly the ARTIC datasets. The dataset is a bit different from usual whole genome shotgun sequencing runs, because we have sequenced amplicons. For Nanopore reads, we have quite short reads that don't overlap by much and are not evenly distributed across the genome. Let's see, if this gets us into trouble.

Of course, in a real world use case, you can't design amplicons before you have the full genome sequence, so you would usually start with whole genome shotgun sequencing...

Canu is a fork of the Celera Assembler, designed for high-noise single-molecule sequencing (such as the PacBio RS II/Sequel or Oxford Nanopore MinION). Documentation can be found here: http://canu.readthedocs.io/en/latest/

Canu is a hierarchical assembly pipeline which runs in four steps: - Detect overlaps in high-noise sequences using MHAP - Generate corrected sequence consensus - Trim corrected sequences - Assemble trimmed corrected sequences

Get a usage message of canu on how to use the assembler:

```
canu --help

usage:   canu [-version] [-citation] \
              [-correct | -trim | -assemble | -trim-assemble] \
              [-s <assembly-specifications-file>] \
               -p <assembly-prefix> \
               -d <assembly-directory> \
               genomeSize=<number>[g|m|k] \
              [other-options] \
              [-pacbio-raw |
               -pacbio-corrected |
               -nanopore-raw |
               -nanopore-corrected] file1 file2 ...

example: canu -d run1 -p godzilla genomeSize=1g -nanopore-raw reads/*.fasta.gz


  To restrict canu to only a specific stage, use:
    -correct       - generate corrected reads
    -trim          - generate trimmed reads
    -assemble      - generate an assembly
    -trim-assemble - generate trimmed reads and then assemble them

  The assembly is computed in the -d <assembly-directory>, with output files named
  using the -p <assembly-prefix>.  This directory is created if needed.  It is not
  possible to run multiple assemblies in the same directory.

  The genome size should be your best guess of the haploid genome size of what is␣
↪being
  assembled.  It is used primarily to estimate coverage in reads, NOT as the desired
  assembly size.  Fractional values are allowed: '4.7m' equals '4700k' equals '4700000
↪'

  Some common options:
    useGrid=string
      - Run under grid control (true), locally (false), or set up for grid control
        but don't submit any jobs (remote)
    rawErrorRate=fraction-error
      - The allowed difference in an overlap between two raw uncorrected reads.  For␣
↪lower
        quality reads, use a higher number.  The defaults are 0.300 for PacBio reads␣
↪and
        0.500 for Nanopore reads.
    correctedErrorRate=fraction-error
      - The allowed difference in an overlap between two corrected reads.  Assemblies␣
↪of
        low coverage or data with biological differences will benefit from a slight␣
↪increase
        in this.  Defaults are 0.045 for PacBio reads and 0.144 for Nanopore reads.
    gridOptions=string
      - Pass string to the command used to submit jobs to the grid.  Can be used to␣
↪set
        maximum run time limits.  Should NOT be used to set memory limits; Canu will␣
↪do
        that for you.
    minReadLength=number
      - Ignore reads shorter than 'number' bases long.  Default: 1000.
```

(continues on next page)

(continued from previous page)

```
    minOverlapLength=number
      - Ignore read-to-read overlaps shorter than 'number' bases long.  Default: 500.
  A full list of options can be printed with '-options'.  All options can be supplied␣
↪in
  an optional sepc file with the -s option.

  Reads can be either FASTA or FASTQ format, uncompressed, or compressed with gz, bz2␣
↪or xz.
  Reads are specified by the technology they were generated with:
    -pacbio-raw         <files>
    -pacbio-corrected   <files>
    -nanopore-raw       <files>
    -nanopore-corrected <files>
```

You can run the complete assembly in one step only. We will run the assembly in two steps: (1) Error correction (2) Trimming and Assembly

After the Error correction, we will generate an error profile as for the raw reads for comparison.

Generate corrected reads

Task: Run the error correction with canu. First of all, create a directory, for your assemblies:

```
mkdir ~/workdir/assembly/
```

Then run:

```
canu -correct
```

with the appropriate parameters, which are:

```
-nanopore-raw <fastq file with reads>
-d <directory where the assembly should be stored>
-p assembly (this will be the prefix for your assembly files)
```

Note: Use the **small** read files as input:

```
~/workdir/data_artic/basecall_small_porechopped.fastq.gz
```

The output directory should be named:

```
~/workdir/assembly/small_correct/
```

In addition, we need some further parameters:

```
useGrid=false (we don't have a cluster)
minReadLength=<minimum read length>
minOverlapLength=<minimum overlap length>
genomeSize=<size of the target genome, i.e. 50k>
```

Choose minReadLength, minOverlapLength and genomeSize to our needs, if you are unsure, what to set here, have a look in the read and mapping statistics again.

If you are stuck for too long, check out the next page for help, but try your best to get the assembly running, before you do that.

References

Canu https://github.com/marbl/canu

1.6.2 Assembly with canu(2)

Generate corrected reads

The command to run the canu correction is:

```
canu -correct
```

with the following parameters:

What?	parameter	Our value
The input read file	-nanopore-raw	~/workdir/data_artic/basecall_small_porechopped.fastq.gz
The output directory	-d	~/workdir/assembly/small_correct
Prefix for output files	-p	assembly
Use a grid engine	useGrid	false
Genome Size	genomeSize	30k
Minimum Read Length	minReadLength	300
Minimum Overlap Length	minOverlapLength	20 or try out different value
Optional: Coverage of corrected reads	corOutCoverage	something smaller than our coverage (~600)
Optional: Min coverage for corrected reads	corMinCoverage	0 to get all
Optional: Correction Sensitivity	corMhapSensitivity	normal

The corOutCoverage parameter defines to which coverage the reads are corrected, longest reads are corrected first. It is advisable to set this parameter high, to get more sequences into the assembly. corMinCoverage set to low value, will report low covered reads as well and corMhapSensitivity=normal is advised for higher coverage.

The complete command is:

```
canu -correct -d ~/workdir/assembly/small_correct -p assembly useGrid=false -nanopore-
↪raw ~/workdir/data_artic/basecall_small_porechopped.fastq.gz genomeSize=30k␣
↪minReadLength=300 minOverlapLength=20
```

Get error statistics

Let's get some error statistics for the corrected reads. Map the corrected reads to the Wuhan reference:

```
minimap2 -a ~/workdir/wuhan.fasta ~/workdir/assembly/small_correct/assembly.
↪correctedReads.fasta.gz | samtools view -b - | samtools sort - > ~/workdir/assembly/
↪small_correct/corrected_reads_vs_wuhan.sorted.bam
```

Then run qualimap:

```
qualimap bamqc -bam ~/workdir/assembly/small_correct/corrected_reads_vs_wuhan.sorted.
↪bam -nw 5000 -nt 14 -c -outdir ~/workdir/assembly/small_correct/qualimap/
```

Then open the results in a web browser:

```
firefox ~/workdir/assembly/small_correct/qualimap/qualimapReport.html
```

Inspect the results, how much did our error rate decrease?

Generate and assemble trimmed reads

The trimming stage identifies unsupported regions in the input and trims or splits reads to their longest supported range. The assembly stage makes a final pass to identify sequencing errors; constructs the best overlap graph (BOG); and outputs contigs, an assembly graph, and summary statistics.

Now run the trimming and assembly step using the following command:

```
canu -trim-assemble
```

You need to define the following parameters:

```
-nanopore-corrected <corrected reads file>
```

The output directory should be named:

```
~/workdir/assembly/small_assembly/
```

In addition, we need some further parameters:

```
useGrid=false (we don't have a cluster)
minReadLength=<minimum read length>
minOverlapLength=<minimum overlap length>
genomeSize=<size of the target genome, i.e. 50k>
```

Use the same parameters as before (although you could use different settings here).

Go to the next page, if you need help.

References

Canu https://github.com/marbl/canu

Minimap2 https://github.com/lh3/minimap2

QualiMap http://qualimap.bioinfo.cipf.es/doc_html/index.html

samtools http://www.htslib.org

1.6.3 Assembly with canu(3)

Run trimming and assembly

The command to run the canu trimming and assembly is:

```
canu -trim-assemble
```

with the following parameters:

What?	parameter	Our value
The input read file	-nanopore-corrected	~/workdir/assembly/small_correct/assembly.correctedReads.fasta.gz
The output directory	-d	~/workdir/assembly/small_assembly
Prefix for output files	-p	assembly
Use a grid engine	useGrid	false
Genome Size	genomeSize	30k
Minimum Read Length	minReadLength	300
Minimum Overlap Length	minOverlapLength	20 or try out different value
Optional: Coverage of corrected reads	corOutCoverage	something smaller than our coverage (~600)
Optional: Min coverage for corrected reads	corMinCoverage	0 to get all
Optional: Correction Sensitivity	corMhapSensitivity	normal

The corOutCoverage parameter defines to which coverage the reads are corrected, longest reads are corrected first. It is advisable to set this parameter high, to get more sequences into the assembly. corMinCoverage set to low value, will report low covered reads as well and corMhapSensitivity=normal is advised for higher coverage.

The complete command is:

```
canu -trim-assemble -d ~/workdir/assembly/small_assembly -p assembly useGrid=false -
↪nanopore-corrected ~/workdir/assembly/small_correct/assembly.correctedReads.fasta.
↪gz genomeSize=30k minReadLength=300 minOverlapLength=20
```

In the next step, we will evaluate our assembly.

References

Canu https://github.com/marbl/canu

Minimap2 https://github.com/lh3/minimap2

BWA http://bio-bwa.sourceforge.net/

samtools http://www.htslib.org

1.6.4 Assembly evaluation with QUAST

We will evaluate our assembly with the tool QUAST.

QUAST stands for QUality ASsessment Tool. The tool evaluates genome assemblies by computing various metrics. You can find all project news and the latest version of the tool at sourceforge. QUAST utilizes MUMmer, GeneMarkS, GeneMark-ES, GlimmerHMM, and GAGE.

Here is the usage for `quast.py`:

```
Usage: python /usr/local/bin/quast.py [options] <files_with_contigs>

Options:
-o  --output-dir  <dirname>       Directory to store all result files [default: quast_
↪results/results_<datetime>]
```

(continues on next page)

(continued from previous page)

```
-r                <filename>      Reference genome file
-g  --features [type:]<filename>  File with genomic feature coordinates in the␣
↪reference (GFF, BED, NCBI or TXT)
                                  Optional 'type' can be specified for extracting␣
↪only a specific feature type from GFF
-m  --min-contig  <int>           Lower threshold for contig length [default: 500]
-t  --threads     <int>           Maximum number of threads [default: 25% of CPUs]

Advanced options:
-s  --split-scaffolds                 Split assemblies by continuous fragments of N's␣
↪and add such "contigs" to the comparison
-l  --labels "label, label, ..."      Names of assemblies to use in reports, comma-
↪separated. If contain spaces, use quotes
-L                                    Take assembly names from their parent directory␣
↪names
-e  --eukaryote                       Genome is eukaryotic (primarily affects gene␣
↪prediction)
    --fungus                          Genome is fungal (primarily affects gene␣
↪prediction)
    --large                           Use optimal parameters for evaluation of large␣
↪genomes
                                      In particular, imposes '-e -m 3000 -i 500 -x␣
↪7000' (can be overridden manually)
-k  --k-mer-stats                     Compute k-mer-based quality metrics␣
↪(recommended for large genomes)
                                      This may significantly increase memory and time␣
↪consumption on large genomes
    --k-mer-size                      Size of k used in --k-mer-stats [default: 101]
    --circos                          Draw Circos plot
-f  --gene-finding                    Predict genes using GeneMarkS (prokaryotes,␣
↪default) or GeneMark-ES (eukaryotes, use --eukaryote)
    --mgm                             Use MetaGeneMark for gene prediction (instead␣
↪of the default finder, see above)
    --glimmer                         Use GlimmerHMM for gene prediction (instead of␣
↪the default finder, see above)
    --gene-thresholds <int,int,...>   Comma-separated list of threshold lengths of␣
↪genes to search with Gene Finding module
                                      [default: 0,300,1500,3000]
    --rna-finding                     Predict ribosomal RNA genes using Barrnap
-b  --conserved-genes-finding         Count conserved orthologs using BUSCO (only on␣
↪Linux)
    --operons  <filename>             File with operon coordinates in the reference␣
↪(GFF, BED, NCBI or TXT)
    --est-ref-size <int>              Estimated reference size (for computing NGx␣
↪metrics without a reference)
    --contig-thresholds <int,int,...> Comma-separated list of contig length␣
↪thresholds [default: 0,1000,5000,10000,25000,50000]
    --x-for-Nx <int>                  Value of 'x' for Nx, Lx, etc metrics reported␣
↪in addition to N50, L50, etc (0, 100) [default: 90]
-u  --use-all-alignments              Compute genome fraction, # genes, # operons in␣
↪QUAST v1.* style.
                                      By default, QUAST filters Minimap's alignments␣
↪to keep only best ones
-i  --min-alignment <int>             The minimum alignment length [default: 65]
    --min-identity <float>            The minimum alignment identity (80.0, 100.0)␣
↪[default: 95.0]
-a  --ambiguity-usage <none|one|all>  Use none, one, or all alignments of a contig␣
↪when all of them
```

(continues on next page)

(continued from previous page)

```
                                        are almost equally good (see --ambiguity-score)␣
↪[default: one]
    --ambiguity-score <float>           Score S for defining equally good alignments of␣
↪a single contig. All alignments are sorted
                                        by decreasing LEN * IDY% value. All alignments␣
↪with LEN * IDY% < S * best(LEN * IDY%) are
                                        discarded. S should be between 0.8 and 1.0␣
↪[default: 0.99]
    --strict-NA                         Break contigs in any misassembly event when␣
↪compute NAx and NGAx.
                                        By default, QUAST breaks contigs only by␣
↪extensive misassemblies (not local ones)
-x  --extensive-mis-size  <int>         Lower threshold for extensive misassembly size.␣
↪All relocations with inconsistency
                                        less than extensive-mis-size are counted as␣
↪local misassemblies [default: 1000]
    --scaffold-gap-max-size  <int>      Max allowed scaffold gap length difference. All␣
↪relocations with inconsistency
                                        less than scaffold-gap-size are counted as␣
↪scaffold gap misassemblies [default: 10000]
    --unaligned-part-size  <int>        Lower threshold for detecting partially␣
↪unaligned contigs. Such contig should have
                                        at least one unaligned fragment >= the␣
↪threshold [default: 500]
    --skip-unaligned-mis-contigs        Do not distinguish contigs with >= 50%␣
↪unaligned bases as a separate group
                                        By default, QUAST does not count misassemblies␣
↪in them
    --fragmented                        Reference genome may be fragmented into small␣
↪pieces (e.g. scaffolded reference)
    --fragmented-max-indent  <int>      Mark translocation as fake if both alignments␣
↪are located no further than N bases
                                        from the ends of the reference fragments␣
↪[default: 85]
                                        Requires --fragmented option
    --upper-bound-assembly              Simulate upper bound assembly based on the␣
↪reference genome and reads
    --upper-bound-min-con  <int>        Minimal number of 'connecting reads' needed for␣
↪joining upper bound contigs into a scaffold
                                        [default: 2 for mate-pairs and 1 for long reads]
    --est-insert-size  <int>            Use provided insert size in upper bound␣
↪assembly simulation [default: auto detect from reads or 255]
    --plots-format  <str>               Save plots in specified format [default: pdf].
                                        Supported formats: emf, eps, pdf, png, ps, raw,␣
↪rgba, svg, svgz
    --memory-efficient                  Run everything using one thread, separately per␣
↪each assembly.
                                        This may significantly reduce memory␣
↪consumption on large genomes
    --space-efficient                   Create only reports and plots files. Aux files␣
↪including .stdout, .stderr, .coords will not be created.
                                        This may significantly reduce space consumption␣
↪on large genomes. Icarus viewers also will not be built
-1  --pe1     <filename>                File with forward paired-end reads (in FASTQ␣
↪format, may be gzipped)
-2  --pe2     <filename>                File with reverse paired-end reads (in FASTQ␣
↪format, may be gzipped)
```

(continues on next page)

(continued from previous page)

```
    --pe12    <filename>              File with interlaced forward and reverse paired-
↪end reads. (in FASTQ format, may be gzipped)
    --mp1     <filename>              File with forward mate-pair reads (in FASTQ␣
↪format, may be gzipped)
    --mp2     <filename>              File with reverse mate-pair reads (in FASTQ␣
↪format, may be gzipped)
    --mp12    <filename>              File with interlaced forward and reverse mate-
↪pair reads (in FASTQ format, may be gzipped)
    --single  <filename>              File with unpaired short reads (in FASTQ format,
↪ may be gzipped)
    --pacbio     <filename>           File with PacBio reads (in FASTQ format, may be␣
↪gzipped)
    --nanopore   <filename>           File with Oxford Nanopore reads (in FASTQ␣
↪format, may be gzipped)
    --ref-sam <filename>              SAM alignment file obtained by aligning reads␣
↪to reference genome file
    --ref-bam <filename>              BAM alignment file obtained by aligning reads␣
↪to reference genome file
    --sam     <filename,filename,...> Comma-separated list of SAM alignment files␣
↪obtained by aligning reads to assemblies
                                      (use the same order as for files with contigs)
    --bam     <filename,filename,...> Comma-separated list of BAM alignment files␣
↪obtained by aligning reads to assemblies
                                      (use the same order as for files with contigs)
                                      Reads (or SAM/BAM file) are used for structural␣
↪variation detection and
                                      coverage histogram building in Icarus
    --sv-bedpe  <filename>            File with structural variations (in BEDPE␣
↪format)

Speedup options:
    --no-check                        Do not check and correct input fasta files. Use␣
↪at your own risk (see manual)
    --no-plots                        Do not draw plots
    --no-html                         Do not build html reports and Icarus viewers
    --no-icarus                       Do not build Icarus viewers
    --no-snps                         Do not report SNPs (may significantly reduce␣
↪memory consumption on large genomes)
    --no-gc                           Do not compute GC% and GC-distribution
    --no-sv                           Do not run structural variation detection (make␣
↪sense only if reads are specified)
    --no-gzip                         Do not compress large output files
    --no-read-stats                   Do not align reads to assemblies
                                      Reads will be aligned to reference and used for␣
↪coverage analysis,
                                      upper bound assembly simulation, and structural␣
↪variation detection.
                                      Use this option if you do not need read␣
↪statistics for assemblies.
    --fast                            A combination of all speedup options except --
↪no-check

Other:
    --silent                          Do not print detailed information about each␣
↪step to stdout (log file is not affected)
    --test                            Run QUAST on the data from the test_data folder,
↪ output to quast_test_output
```

(continues on next page)

(continued from previous page)

```
    --test-sv                          Run QUAST with structural variants detection on␣
↪the data from the test_data folder,
                                       output to quast_test_output
-h  --help                             Print full usage message
-v  --version                          Print version
```

You can start the tool with:

```
quast.py
```

To call `quast.py` we have to provide a reference genome and one or more assemblies. The reference is the wuhan reference. Find the appropriate parameters in the usage and use:

```
-t <number of threads>
```

in addition.

The output should be stored in:

```
~/workdir/assembly/small_assembly/quast/
```

When you are done (or stuck) go to the next page.

References

quast http://sourceforge.net/projects/quast

1.6.5 Assembly evaluation with QUAST(2)

We run:

```
quast.py
```

with the following parameters:

The complete commandline is:

```
quast.py -t 14 -o ~/workdir/assembly/small_assembly/quast/ -r ~/workdir/wuhan.fasta ~/
↪workdir/assembly/small_assembly/assembly.contigs.fasta
```

QUAST generates HTML reports including a number of interactive graphics. To access these reports, open them using a file browser:

```
firefox ~/workdir/assembly/small_assembly/quast/report.html
```

Inspect the results - are you satisfied with the assembly?

References

quast http://sourceforge.net/projects/quast

1.6.6 Assembly Graph inspection with Bandage

This did'nt work out very well, let's have a look on the assembly graph with Bandage.

Bandage is a program for visualising de novo assembly graphs. By displaying connections which are not present in the contigs file, Bandage opens up new possibilities for analysing *de novo* assemblies.

You can start Bandage with:

```
Bandage
```

and load the following file:

```
~/workdir/assembly/small_assembly/assembly.contigs.gfa
```

and click on "Draw Graph" This is the assembly graph of our Nanopore Assembly. You can BLAST your contigs vs each other to identify further assembly problems by clicking "Create/View BLAST search".

There seem to be problems with overlaps in the assembly and also unequally distributed coverage. canu does not seem to handle our data well. If you want, you can try to get a better assembly. Other parameters you might want to change are mentioned in the canu part, others might be:

```
correctedErrorRate (default 0.144)
utgErrorRate (default 0.144)
```

We will now use a different dataset to get a complete assembly.

References

Bandage https://rrwick.github.io/Bandage/

1.6.7 Get and inspect the WGS dataset

Get the data

The dataset we are going to use is a whole genome shotgun project from an outbreak in Hongkong.

The dataset is located in our object store. Download it with wget:

```
cd ~/workdir
mkdir data_wgs
cd data_wgs
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/Cov2_HK_
↪WGS_small.fastq.gz
```

Remove adapters

Run porechop on the dataset (try to get the command right on your own):

```
porechop -t 14 -v 2 -i ~/workdir/data_wgs/Cov2_HK_WGS_small.fastq.gz -o ~/workdir/
↪data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz | tee ~/workdir/data_wgs/porechop.
↪log
```

We save the output with `tee` in `porechop.log` if you want to reinspect it.

Map the data and inspect with GenomeView

Map the data to the Wuhan reference:

```
minimap2 -a -t 14 ~/workdir/wuhan.fasta ~/workdir/data_wgs/Cov2_HK_WGS_small_
↪porechopped.fastq.gz | samtools view -b - | samtools sort - > ~/workdir/mappings/
↪Cov2_HK_WGS_small_porechopped_vs_wuhan.sorted.bam
```

We also map the small dataset to the Wuhan reference since we haven't done that yet:

```
minimap2 -t 14 -x map-ont -a  ~/workdir/wuhan.fasta ~/workdir/data_artic/basecall_
↪small_porechopped.fastq.gz | samtools view -b - | samtools sort - > ~/workdir/
↪mappings/basecall_small_porechopped_vs_wuhan.sorted.bam
```

Create the indizes:

```
samtools index ~/workdir/mappings/Cov2_HK_WGS_small_porechopped_vs_wuhan.sorted.bam
samtools index ~/workdir/mappings/basecall_small_porechopped_vs_wuhan.sorted.bam
```

Load GenomeView with:

```
java -jar ~/genomeview-N42.jar
```

Load the Wuhan reference and the mappings and look at the data - is it more equally distributed?

In the next step, we perform the assembly with canu and the WGS dataset.

References

Porechop https://github.com/rrwick/Porechop

Minimap2 https://github.com/lh3/minimap2

samtools http://www.htslib.org

GenomeView https://genomeview.org/

1.6.8 Assembly of WGS dataset

We will now assemble the WGS dataset.

Run assembly

The command to run the canu assembly is:

```
canu
```

Without -correct or -trim-assemble, the complete assembly will run in one step.

Use the appropriate parameters, which are:

```
-nanopore-raw <fastq file with reads>
-d <directory where the assembly should be stored>
-p assembly (this will be the prefix for your assembly files)
```

Make sure, to use the porechopped reads as input!

The output directory should be named:

```
~/workdir/assembly/assembly_wgs/
```

In addition, we need some further parameters:

```
useGrid=false (we don't have a cluster)
minReadLength=<minimum read length>
minOverlapLength=<minimum overlap length>
genomeSize=<size of the target genome, i.e. 50k>
```

You already know these parameters from our previous assembly. We don't have amplicon data now, but the reads are still quite short (for Nanopore reads). The default minReadLength is 1000, this might remove quite a bit of our reads, so choose something smaller. minOverlapLength does not need to be as small as for the amplicon dataset, since we have larger overlaps now. It still needs to be smaller than the minReadLength.

Try parameters, that you think, make sense. If your assembly sucks - try again. Or check the next page for parameters that should work.

References

Canu https://github.com/marbl/canu

1.6.9 Assembly with WGS dataset(2)

Generate corrected reads

The command to run the canu correction is:

```
canu
```

with the following parameters:

What?	parameter	Our value
The input read file	-nanopore-raw	~/workdir/data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz
The output directory	-d	~/workdir/assembly/assembly_wgs/
Prefix for output files	-p	assembly
Use a grid engine	useGrid	false
Genome Size	genomeSize	30k
Minimum Read Length	minReadLength	300
Minimum Overlap Length	minOverlapLength	20

The complete command is:

```
canu -d ~/workdir/assembly/assembly_wgs/ -p assembly useGrid=false -nanopore-raw ~/
↪workdir/data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz genomeSize=30k␣
↪minReadLength=300 minOverlapLength=20
```

References

Canu https://github.com/marbl/canu

1.6.10 WGS Assembly evaluation with QUAST

We will evaluate our WGS assembly with the tool QUAST.

QUAST stands for QUality ASsessment Tool. The tool evaluates genome assemblies by computing various metrics. You can find all project news and the latest version of the tool at sourceforge. QUAST utilizes MUMmer, GeneMarkS, GeneMark-ES, GlimmerHMM, and GAGE.

Here is the usage for `quast.py`:

```
Usage: python /usr/local/bin/quast.py [options] <files_with_contigs>

Options:
-o  --output-dir  <dirname>       Directory to store all result files [default: quast_
↪results/results_<datetime>]
-r                <filename>      Reference genome file
-g  --features [type:]<filename>  File with genomic feature coordinates in the␣
↪reference (GFF, BED, NCBI or TXT)
                                  Optional 'type' can be specified for extracting␣
↪only a specific feature type from GFF
-m  --min-contig  <int>           Lower threshold for contig length [default: 500]
-t  --threads     <int>           Maximum number of threads [default: 25% of CPUs]

Advanced options:
-s  --split-scaffolds                 Split assemblies by continuous fragments of N's␣
↪and add such "contigs" to the comparison
-l  --labels "label, label, ..."      Names of assemblies to use in reports, comma-
↪separated. If contain spaces, use quotes
-L                                    Take assembly names from their parent directory␣
↪names
-e  --eukaryote                       Genome is eukaryotic (primarily affects gene␣
↪prediction)
    --fungus                          Genome is fungal (primarily affects gene␣
↪prediction)
    --large                           Use optimal parameters for evaluation of large␣
↪genomes
                                      In particular, imposes '-e -m 3000 -i 500 -x␣
↪7000' (can be overridden manually)
-k  --k-mer-stats                     Compute k-mer-based quality metrics␣
↪(recommended for large genomes)
                                      This may significantly increase memory and time␣
↪consumption on large genomes
    --k-mer-size                      Size of k used in --k-mer-stats [default: 101]
    --circos                          Draw Circos plot
-f  --gene-finding                    Predict genes using GeneMarkS (prokaryotes,␣
↪default) or GeneMark-ES (eukaryotes, use --eukaryote)
    --mgm                             Use MetaGeneMark for gene prediction (instead␣
↪of the default finder, see above)
    --glimmer                         Use GlimmerHMM for gene prediction (instead of␣
↪the default finder, see above)
    --gene-thresholds <int,int,...>   Comma-separated list of threshold lengths of␣
↪genes to search with Gene Finding module
                                      [default: 0,300,1500,3000]
```

(continues on next page)

(continued from previous page)

```
    --rna-finding                       Predict ribosomal RNA genes using Barrnap
-b  --conserved-genes-finding           Count conserved orthologs using BUSCO (only on
↪Linux)
    --operons  <filename>               File with operon coordinates in the reference
↪(GFF, BED, NCBI or TXT)
    --est-ref-size <int>                Estimated reference size (for computing NGx
↪metrics without a reference)
    --contig-thresholds <int,int,...> Comma-separated list of contig length
↪thresholds [default: 0,1000,5000,10000,25000,50000]
    --x-for-Nx <int>                    Value of 'x' for Nx, Lx, etc metrics reported
↪in addition to N50, L50, etc (0, 100) [default: 90]
-u  --use-all-alignments                Compute genome fraction, # genes, # operons in
↪QUAST v1.* style.
                                        By default, QUAST filters Minimap's alignments
↪to keep only best ones
-i  --min-alignment <int>               The minimum alignment length [default: 65]
    --min-identity <float>              The minimum alignment identity (80.0, 100.0)
↪[default: 95.0]
-a  --ambiguity-usage <none|one|all>  Use none, one, or all alignments of a contig
↪when all of them
                                        are almost equally good (see --ambiguity-score)
↪[default: one]
    --ambiguity-score <float>           Score S for defining equally good alignments of
↪a single contig. All alignments are sorted
                                        by decreasing LEN * IDY% value. All alignments
↪with LEN * IDY% < S * best(LEN * IDY%) are
                                        discarded. S should be between 0.8 and 1.0
↪[default: 0.99]
    --strict-NA                         Break contigs in any misassembly event when
↪compute NAx and NGAx.
                                        By default, QUAST breaks contigs only by
↪extensive misassemblies (not local ones)
-x  --extensive-mis-size  <int>         Lower threshold for extensive misassembly size.
↪All relocations with inconsistency
                                        less than extensive-mis-size are counted as
↪local misassemblies [default: 1000]
    --scaffold-gap-max-size  <int>      Max allowed scaffold gap length difference. All
↪relocations with inconsistency
                                        less than scaffold-gap-size are counted as
↪scaffold gap misassemblies [default: 10000]
    --unaligned-part-size  <int>        Lower threshold for detecting partially
↪unaligned contigs. Such contig should have
                                        at least one unaligned fragment >= the
↪threshold [default: 500]
    --skip-unaligned-mis-contigs        Do not distinguish contigs with >= 50%
↪unaligned bases as a separate group
                                        By default, QUAST does not count misassemblies
↪in them
    --fragmented                        Reference genome may be fragmented into small
↪pieces (e.g. scaffolded reference)
    --fragmented-max-indent  <int>      Mark translocation as fake if both alignments
↪are located no further than N bases
                                        from the ends of the reference fragments
↪[default: 85]
                                        Requires --fragmented option
    --upper-bound-assembly              Simulate upper bound assembly based on the
↪reference genome and reads
```

(continues on next page)

(continued from previous page)

```
    --upper-bound-min-con  <int>      Minimal number of 'connecting reads' needed for␣
↪joining upper bound contigs into a scaffold
                                      [default: 2 for mate-pairs and 1 for long reads]
    --est-insert-size  <int>          Use provided insert size in upper bound␣
↪assembly simulation [default: auto detect from reads or 255]
    --plots-format  <str>             Save plots in specified format [default: pdf].
                                      Supported formats: emf, eps, pdf, png, ps, raw,␣
↪rgba, svg, svgz
    --memory-efficient                Run everything using one thread, separately per␣
↪each assembly.
                                      This may significantly reduce memory␣
↪consumption on large genomes
    --space-efficient                 Create only reports and plots files. Aux files␣
↪including .stdout, .stderr, .coords will not be created.
                                      This may significantly reduce space consumption␣
↪on large genomes. Icarus viewers also will not be built
-1  --pe1     <filename>              File with forward paired-end reads (in FASTQ␣
↪format, may be gzipped)
-2  --pe2     <filename>              File with reverse paired-end reads (in FASTQ␣
↪format, may be gzipped)
    --pe12    <filename>              File with interlaced forward and reverse paired-
↪end reads. (in FASTQ format, may be gzipped)
    --mp1     <filename>              File with forward mate-pair reads (in FASTQ␣
↪format, may be gzipped)
    --mp2     <filename>              File with reverse mate-pair reads (in FASTQ␣
↪format, may be gzipped)
    --mp12    <filename>              File with interlaced forward and reverse mate-
↪pair reads (in FASTQ format, may be gzipped)
    --single  <filename>              File with unpaired short reads (in FASTQ format,
↪ may be gzipped)
    --pacbio     <filename>           File with PacBio reads (in FASTQ format, may be␣
↪gzipped)
    --nanopore   <filename>           File with Oxford Nanopore reads (in FASTQ␣
↪format, may be gzipped)
    --ref-sam <filename>              SAM alignment file obtained by aligning reads␣
↪to reference genome file
    --ref-bam <filename>              BAM alignment file obtained by aligning reads␣
↪to reference genome file
    --sam     <filename,filename,...> Comma-separated list of SAM alignment files␣
↪obtained by aligning reads to assemblies
                                      (use the same order as for files with contigs)
    --bam     <filename,filename,...> Comma-separated list of BAM alignment files␣
↪obtained by aligning reads to assemblies
                                      (use the same order as for files with contigs)
                                      Reads (or SAM/BAM file) are used for structural␣
↪variation detection and
                                      coverage histogram building in Icarus
    --sv-bedpe  <filename>            File with structural variations (in BEDPE␣
↪format)

Speedup options:
    --no-check                        Do not check and correct input fasta files. Use␣
↪at your own risk (see manual)
    --no-plots                        Do not draw plots
    --no-html                         Do not build html reports and Icarus viewers
    --no-icarus                       Do not build Icarus viewers
    --no-snps                         Do not report SNPs (may significantly reduce␣
↪memory consumption on large genomes)
```

(continues on next page)

(continued from previous page)

```
    --no-gc                             Do not compute GC% and GC-distribution
    --no-sv                             Do not run structural variation detection (make␣
↪sense only if reads are specified)
    --no-gzip                           Do not compress large output files
    --no-read-stats                     Do not align reads to assemblies
                                        Reads will be aligned to reference and used for␣
↪coverage analysis,
                                        upper bound assembly simulation, and structural␣
↪variation detection.
                                        Use this option if you do not need read␣
↪statistics for assemblies.
    --fast                              A combination of all speedup options except --
↪no-check

Other:
    --silent                            Do not print detailed information about each␣
↪step to stdout (log file is not affected)
    --test                              Run QUAST on the data from the test_data folder,
↪ output to quast_test_output
    --test-sv                           Run QUAST with structural variants detection on␣
↪the data from the test_data folder,
                                        output to quast_test_output
-h  --help                              Print full usage message
-v  --version                           Print version
```

You can start the tool with:

```
quast.py
```

To call `quast.py` we have to provide a reference genome and one or more assemblies. The reference is the wuhan reference. Find the appropriate parameters in the usage and use:

```
-t <number of threads>
```

in addition.

The output should be stored in:

```
~/workdir/assembly/assembly_wgs/quast
```

When you are done (or stuck) go to the next page.

References

quast http://sourceforge.net/projects/quast

1.6.11 WGS Assembly evaluation with QUAST(2)

We run:

```
quast.py
```

with the following parameters:

What?	parameter	Our value
The input assembly	positional	~/workdir/assembly/assembly_wgs/assembly.contigs.fasta/
The output directory	-o	~/workdir/assembly/assembly_wgs/quast/
The number of threads to be used	-t	14

The complete commandline is:

```
quast.py -t 14 -o ~/workdir/assembly/assembly_wgs/quast/ -r ~/workdir/wuhan.fasta ~/
↪workdir/assembly/assembly_wgs/assembly.contigs.fasta
```

QUAST generates HTML reports including a number of interactive graphics. To access these reports, open them using a file browser:

```
firefox ~/workdir/assembly/assembly_wgs/quast/report.html
```

Inspect the results - are you satisfied with the assembly now?

References

quast http://sourceforge.net/projects/quast

1.6.12 WGS Assembly Graph inspection with Bandage

Let's have a look on the assembly graph with Bandage.

Bandage is a program for visualising de novo assembly graphs. By displaying connections which are not present in the contigs file, Bandage opens up new possibilities for analysing *de novo* assemblies.

You can start Bandage with:

```
Bandage
```

and load the following file:

```
~/workdir/assembly/assembly_wgs/assembly.contigs.gfa
```

and click on "Draw Graph" This is the assembly graph of our Nanopore Assembly. You can BLAST your contigs vs each other to identify further assembly problems by clicking "Create/View BLAST search".

References

Bandage https://rrwick.github.io/Bandage/

1.6.13 Quality control by mapping

In this part of the tutorial we will look at the assemblies by mapping the contigs of our first assembly to the reference genome using Minimap2, samtools and GenomeView.

Map the data and inspect with GenomeView

Your task is, to map the WGS assembly to the Wuhan reference, create a sorted and indexed bam file and view the results in GenomeView.

The mapping result file (sorted bam) should be named:

```
~/workdir/mappings/assembly_wgs_vs_wuhan.sorted.bam
```

You already did that a few times, so we are quite confident, you can do that without help. If not - check out the next page.

References

Minimap2 https://github.com/lh3/minimap2

samtools http://www.htslib.org

GenomeView https://genomeview.org/

1.6.14 Quality control by mapping(2)

Map the data to the Wuhan reference:

```
minimap2 -a -t 14 ~/workdir/wuhan.fasta ~/workdir/assembly/assembly_wgs/assembly.
↪contigs.fasta | samtools view -b - | samtools sort - > ~/workdir/mappings/assembly_
↪wgs_vs_wuhan.sorted.bam
```

Create the index:

```
samtools index ~/workdir/mappings/assembly_wgs_vs_wuhan.sorted.bam
```

Load GenomeView with:

```
java -jar ~/genomeview-N42.jar
```

Load the wuhan reference and the mapping and look at the data - is it more equally distributed?

In the next step, we perform the assembly with canu and the WGS dataset.References

Minimap2 https://github.com/lh3/minimap2

samtools http://www.htslib.org

IGV http://www.broadinstitute.org/igv/

1.7 Assembly polishing

There are two general strategies for polishing: 1. Polish with Illumina data 2. Polishing with Nanopore data

For the first the tool `Pilon` can be used. Since we don't have Illumina data for our WGS assembly, we can't do that here. We have Illumina data, for the ARTIC datasets, so we will repeat this part on the last day, where we generate consensus sequences using the ARTIC pipeline.

However, we can do the second polishing. There are 2 widely used tools to do that:

1. Nanopolish (works on raw fast5 files)
2. medaka (works with fastq files)

We only have fastq files, so we need to stick to the medaka polishing, which works better in practice most of the times anyway.

We have used nanopolish in previous courses. If you are interested in an example call, check out the old documentation:

https://denbi-nanopore-training-course.readthedocs.io/en/latest/polishing/nanopolish/index.html

1.7.1 Assembly polishing with racon and medaka

We are going to polish our assembly using racon and medaka now.

Polishing with Racon

Racon is intended as a standalone consensus module to correct raw contigs generated by rapid assembly methods which do not include a consensus step. The goal of Racon is to generate genomic consensus which is of similar or better quality compared to the output generated by assembly methods which employ both error correction and consensus steps, while providing a speedup of several times compared to those methods. It supports data produced by both Pacific Biosciences and Oxford Nanopore Technologies.

Racon can be used as a polishing tool after the assembly with either Illumina data or data produced by third generation of sequencing. The type of data inputed is automatically detected.

Racon takes as input only three files: contigs in FASTA/FASTQ format, reads in FASTA/FASTQ format and overlaps/alignments between the reads and the contigs in MHAP/PAF/SAM format. Output is a set of polished contigs in FASTA format printed to stdout. All input files can be compressed with gzip (which will have impact on parsing time).

We are going to use racon to do an initial correction. The medaka documentation advises to do four rounds with racon before polishing with medaka since medaka has been trained with racon polished assemblies. We are only doing one round here.

Mapping of Nanopore reads to the assembly

In order to use racon, we need a mapping of the reads to the assembly. We use `minimap2` for this task.

Map the data to the assembly:

```
minimap2 -a -t 14 ~/workdir/assembly/assembly_wgs/assembly.contigs.fasta ~/workdir/
↪data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz  > ~/workdir/mappings/Cov2_HK_WGS_
↪small_porechopped_vs_assembly_wgs.sam
```

Run racon

Check the usage of racon:

```
racon --help
usage: racon [options ...] <sequences> <overlaps> <target sequences>

  <sequences>
      input file in FASTA/FASTQ format (can be compressed with gzip)
```

(continues on next page)

(continued from previous page)

```
    containing sequences used for correction
<overlaps>
    input file in MHAP/PAF/SAM format (can be compressed with gzip)
    containing overlaps between sequences and target sequences
<target sequences>
    input file in FASTA/FASTQ format (can be compressed with gzip)
    containing sequences which will be corrected

options:
    -u, --include-unpolished
        output unpolished target sequences
    -f, --fragment-correction
        perform fragment correction instead of contig polishing
        (overlaps file should contain dual/self overlaps!)
    -w, --window-length <int>
        default: 500
        size of window on which POA is performed
    -q, --quality-threshold <float>
        default: 10.0
        threshold for average base quality of windows used in POA
    -e, --error-threshold <float>
        default: 0.3
        maximum allowed error rate used for filtering overlaps
    -m, --match <int>
        default: 5
        score for matching bases
    -x, --mismatch <int>
        default: -4
        score for mismatching bases
    -g, --gap <int>
        default: -8
        gap penalty (must be negative)
    -t, --threads <int>
        default: 1
        number of threads
    --version
        prints the version number
    -h, --help
        prints the usage
```

We need to call:

```
racon
```

with our reads, our mapping (in sam format) and the reference assembly (in that order). We use 14 threads:

```
-t 14
```

And in addition the following parameters:

```
-m 8 -x -6 -g -8 -w 500
```

. . . because they were also used for the training of medaka and we want to have similar error profiles of the draft.

Racon Problems

If you are having trouble running racon and get a "Illegal instruction (core dumped)" message, try reinstalling with the following commands:

```
sudo rm /usr/local/bin/racon
git clone --recursive https://github.com/lbcb-sci/racon.git racon
cd racon
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
cd
rm -rf ~/racon/
```

References

Minimap2 https://github.com/lh3/minimap2

samtools http://www.htslib.org

racon https://github.com/isovic/racon

Polishing with Racon(2)

We call racon:

```
racon
```

with the following parameters:

```
~/workdir/data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz
~/workdir/mappings/Cov2_HK_WGS_small_porechopped_vs_wuhan.sam
~/workdir/wuhan.fasta
```

What?	parameter	Our value
The input read file	positional (1)	~/workdir/data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz
The mapping file	positional (2)	~/workdir/mappings/Cov2_HK_WGS_small_porechopped_vs_assembly_wgs.sam
The reference file	positional (3)	~/workdir/assemmbly/assembly_wgs/assemby.contigs.fasta
Number of threads	-t	14
Racon parameters for medaka	-m 8 -x -6 -g -8 -w 500	
Output file	forward with ">"	~/workdir/assembly/assembly_wgs/racon.fasta

Then we can call racon with our mapping, the read file and the assembly file. We use 14 threads to do this:

```
racon -m 8 -x -6 -g -8 -w 500 -t 14 ~/workdir/data_wgs/Cov2_HK_WGS_small_porechopped.
↪fastq.gz  ~/workdir/mappings/Cov2_HK_WGS_small_porechopped_vs_assembly_wgs.sam ~/
↪workdir/assembly/assembly_wgs/assembly.contigs.fasta > ~/workdir/assembly/assembly_
↪wgs/racon.fasta
```

References

racon https://github.com/isovic/racon

Polishing with medaka

Medaka is a tool to create a consensus sequence of nanopore sequencing data. This task is performed using neural networks applied a pileup of individual sequencing reads against a draft assembly. It outperforms graph-based methods operating on basecalled data, and can be competitive with state-of-the-art signal-based methods whilst being much faster.

In earlier courses, we used nanopolish for polishing but it is outperformed by medaka in both runtime and accuracy.

As input medaka accepts a sorted and indexed BAM mapping file. It requires a draft assembly as a .fasta.

Medaka hast 3 steps / subtools:

```
mini_align (basically runs a minimap2 mapping)
medaka consensus (generates a consensus, you can do that for subparts of the assembly␣
↪to improve runtime)
medaka stitch (to stitch the subparts together, or generate a fasta from the results␣
↪from medaka consensus)
```

However, for smaller assemblies, we can just use `medaka_consensus` that performs all the steps above:

```
medaka 1.2.0
------------

Assembly polishing via neural networks. The input assembly should be
preprocessed with racon.

medaka_consensus [-h] -i <fastx>

    -h  show this help text.
    -i  fastx input basecalls (required).
    -d  fasta input assembly (required).
    -o  output folder (default: medaka).
    -g  don't fill gaps in consensus with draft sequence.
    -m  medaka model, (default: r941_min_high_g360).
        Available: r103_min_high_g345, r103_min_high_g360, r103_prom_high_g360, r103_
↪prom_snp_g3210, r103_prom_variant_g3210, r10_min_high_g303, r10_min_high_g340, r941_
↪min_fast_g303, r941_min_high_g303, r941_min_high_g330, r941_min_high_g340_rle, r941_
↪min_high_g344, r941_min_high_g351, r941_min_high_g360, r941_prom_fast_g303, r941_
↪prom_high_g303, r941_prom_high_g330, r941_prom_high_g344, r941_prom_high_g360, r941_
↪prom_high_g4011, r941_prom_snp_g303, r941_prom_snp_g322, r941_prom_snp_g360, r941_
↪prom_variant_g303, r941_prom_variant_g322, r941_prom_variant_g360.
        Alternatively a .hdf file from 'medaka train'.
    -f  Force overwrite of outputs (default will reuse existing outputs).
    -t  number of threads with which to create features (default: 1).
    -b  batchsize, controls memory use (default: 100).

-i must be specified.
```

We need to define the following parameters:

```
-i <input fastq>
-d <racon reference assembly>
```

(continues on next page)

(continued from previous page)

```
-o <output folder>, should be: ~/workdir/assembly/assembly_wgs/medaka/
-t <threads>
-m <the appropriate medaka model>
```

The model are named with the following scheme:

```
{pore}_{device}_{caller variant}_{caller version}
```

Our pore is r941, the device is MinION (min), we take the high-accuracy model (high), and our guppy version was 4.15. Choose the model, that is closest to that basecaller version.

If you are stuck, get help on the next page.

References

medaka https://github.com/nanoporetech/medaka

Polishing with medaka(2)

We call:

```
medaka_consensus
```

with the following parameters:

What?	parameter	Our value
The input read file	-i	~/workdir/data_wgs/Cov2_HK_WGS_small_porechopped.fastq.gz
The racon polished assembly	-d	~/workdir/assembly/assembly_wgs/racon.fasta
The output directory	-o	~/workdir/assembly/assembly_wgs/medaka
Number of threads	-t	14
The model	-m	r941_min_high_g360

The complete commandline is:

```
medaka_consensus -t 14 -m r941_min_high_g360 -i ~/workdir/data_wgs/Cov2_HK_WGS_small_
↪porechopped.fastq.gz  -d ~/workdir/assembly/assembly_wgs/racon.fasta -o ~/workdir/
↪assembly/assembly_wgs/medaka
```

In a next step, we will use quast to compare our assemblies.

If medaka_consensus fails with an error massage try the following an re-run the command above:

```
pip install tensorflow
```

References

medaka https://github.com/nanoporetech/medaka

Assembly evaluation with quast

As usual, we are going to use quast for assembly evaluation:

```
quast.py -t 14 -o ~/workdir/assembly/assembly_wgs/quast_2/ -r ~/workdir/wuhan.fasta ~/
↪workdir/assembly/assembly_wgs/assembly.contigs.fasta  ~/workdir/assembly/assembly_
↪wgs/racon.fasta  ~/workdir/assembly/assembly_wgs/medaka/consensus.fasta
```

QUAST generates HTML reports including a number of interactive graphics. To access these reports, open a file browser:

```
firefox ~/workdir/assembly/assembly_wgs/quast_2/report.html
```

References

quast http://sourceforge.net/projects/quast

1.8 Recap Day 2 / Preparations Day 3

We have performed an assembly on the ARTIC dataset and since it didn't work well we did another using a WGS dataset which we also polished using racon and medaka.

Today, we are working with the ARTIC datasets again. Please go to your data_artic directory, and download all datasets, untar and remove the tar files:

```
cd ~/workdir/data_artic/

for i in {1..5}
do
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/basecall_
↪0$i.tar.gz
tar -xzvf basecall_0$i.tar.gz
rm basecall_0$i.tar.gz
done
```

We also have some Illumina data fitting to our samples:

```
for i in {1..5}
do
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/Illumina_
↪0$i.tar.gz
tar -xzvf Illumina_0$i.tar.gz
rm Illumina_0$i.tar.gz
done
```

1.9 ARTIC pipeline

In this part of the tutorial we are going to use the ARTIC pipeline to generate consensus sequences from our ARTIC amplicon data. We will then polish the consensus with Illumina data using Pilon. As a final step, we will run Nextstrain with example SARS-Cov2 data and include our own sequences.

1.9.1 ARTIC pipeline

In this part of the tutorial, we are using the ARTIC pipeline to generate a consensus sequence from the Wuhan reference and our amplicon data.

The complete ARTIC bioinformatics SOP can be found here: https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html

Since we already have basecalled and demultiplexed data, we start with the step "Read filtering".

The ARTIC pipeline is installed in a conda environment which we need to activate with:

```
conda activate artic-ncov2019
```

When activated, your shell looks like this:

```
(artic-ncov2019) ubuntu@nanopore-course-2020:~$
```

This indicates, that the artic-ncov2019 environment is active. When no environment is active, your shell indicates that with "(base)" at the start of the line like this:

```
(base) ubuntu@nanopore-course-2020:~$
```

This is for example the case, when you open a new shell. You can also deactivate the active conda environment with:

```
conda deactivate
```

Always make sure the artic-ncov2019 environment is active in this part of the tutorial.

Length filtering

First of all, if not active, activate the artic-ncov2019 conda environment:

```
conda activate artic-ncov2019
```

The we will use the command:

```
artic guppyplex
```

to perform a length filtering on the basecalled data and combine all reads into one single file:

```
usage: artic guppyplex [-h] [-q] --directory directory
                       [--max-length max_length] [--min-length min_length]
                       [--quality quality] [--sample sample]
                       [--skip-quality-check] [--prefix PREFIX]
                       [--output output]

optional arguments:
  -h, --help            show this help message and exit
  -q, --quiet           Do not output warnings to stderr
  --directory directory
                        Basecalled and demultiplexed (guppy) results directory
  --max-length max_length
                        remove reads greater than read length
  --min-length min_length
                        remove reads less than read length
  --quality quality     remove reads against this quality filter
```

(continues on next page)

(continued from previous page)

```
  --sample sample       sampling frequency for random sample of sequence to
                        reduce excess
  --skip-quality-check  Do not filter on quality score (speeds up)
  --prefix PREFIX       Prefix for guppyplex files
  --output output       FASTQ file to write
```

Task: Use `artic guppyplex` to filter for reads with a minimum size of 400 and a maximum size of 700. Your output files should be named:

```
~/workdir/data_artic/basecall_filtered_01.fastq
```

Do the filtering for the first (01) of the 5 datasets only and stick to that dataset for the next parts. When you are quick, you can repeat the procedure for the remaining datasets later.

References

ARTIC bioinformatics SOP https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html

Length filtering (2)

First of all, if not active, activate the artic-ncov2019 conda environment:

```
conda activate artic-ncov2019
```

Then use the command:

```
artic guppyplex
```

with the following parameters:

What?	parameter	Our value
The input directory containing the reads	–directory	~/workdir/data_artic/basecall_01/
The output file	–output	~/workdir/data_artic/basecall_filtered_01.fastq
Minimum read length	–min-length	400
Maximum read length	–max-length	700
(optional) Skip quality check	–skip-quality-check	

Since the quality check has been done along with the basecalling, we can use the flag `--skip-quality-check`. That will improve runtime, but does not really change much.

To perform the filtering for one dataset, we can use the following command:

```
artic guppyplex --skip-quality-check --min-length 400 --max-length 700 --directory ~/
↪workdir/data_artic/basecall_01/ --output ~/workdir/data_artic/basecall_filtered_01.
↪fastq
```

Perform that step for the first (01) dataset only to save time. Do the other datasets later, when there is time left.

If you wanted to do that for all datasaets, you could do that in a loop:

```
for i in {1..5}
do artic guppyplex --skip-quality-check --min-length 400 --max-length 700 --directory␣
↪~/workdir/data_artic/basecall_0$i --output ~/workdir/data_artic/basecall_filtered_0
↪$i.fastq
done
```

In the next step, we use the filtered reads to generate consensus sequences.

References

ARTIC bioinformatics SOP https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html

Generating consensus sequence

First of all, if not active, activate the artic-ncov2019 conda environment:

```
conda activate artic-ncov2019
```

We will now use the artic pipeline to call variants in the Wuhan reference using our amlicon dataset. The command to do that is:

```
artic minion
```

There are two tools, that the pipeline uses to call variants: 1) Nanopolish 2) Medaka

Since Medaka is faster and in our experience more accurate we will use the second option. You need to set the appropriate command line flag to do that.

Check the usage for artic minion:

```
usage: artic minion [-h] [-q] [--medaka] [--minimap2] [--bwa]
                    [--normalise NORMALISE] [--threads THREADS]
                    [--scheme-directory scheme_directory]
                    [--max-haplotypes max_haplotypes] [--read-file read_file]
                    [--fast5-directory FAST5_DIRECTORY]
                    [--sequencing-summary SEQUENCING_SUMMARY]
                    [--skip-nanopolish] [--no-indels] [--dry-run]
                    scheme sample

positional arguments:
  scheme                The name of the scheme.
  sample                The name of the sample.

optional arguments:
  -h, --help            show this help message and exit
  -q, --quiet           Do not output warnings to stderr
  --medaka              Use medaka instead of nanopolish for variants
  --minimap2            Use minimap2 (default)
  --bwa                 Use bwa instead of minimap2
  --normalise NORMALISE
                        Normalise down to moderate coverage to save runtime.
  --threads THREADS     Number of threads
  --scheme-directory scheme_directory
                        Default scheme directory
  --max-haplotypes max_haplotypes
```

(continues on next page)

(continued from previous page)

```
                        max-haplotypes value for nanopolish
  --read-file read_file
                        Use alternative FASTA/FASTQ file to <sample>.fasta
  --fast5-directory FAST5_DIRECTORY
                        FAST5 Directory
  --sequencing-summary SEQUENCING_SUMMARY
                        Path to Guppy sequencing summary
  --skip-nanopolish
  --no-indels
  --dry-run
```

Create a directory for the results and cd into it:

```
mkdir ~/workdir/results_artic/
cd ~/workdir/results_artic/
```

Then run the `artic minion` command using medaka, use 14 threads, you can normalise to 200fold coverage to save runtime if you want. You need to set the correct scheme directory (containing primer sequences), which is:

```
~/artic-ncov2019/primer_schemes
```

And as positional arguments, you need to provide:

```
(1) the exact primer sequences:
nCoV-2019/V3

(2) The samplename:
barcode_01
```

Perform that step for the first (01) dataset only to save time. Do the other datasets later, when there is time left.

If you are stuck, get help on the next page.

References

ARTIC bioinformatics SOP https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html

Generating consensus sequence (2)

First of all, if not active, activate the artic-ncov2019 conda environment:

```
conda activate artic-ncov2019
```

Then use the command:

```
artic minion
```

with the following parameters:

What?	parameter	Our value
Use medaka	–medaka	
The directory containing primer schemes	–scheme-directory	~/artic-ncov2019/primer_schemes
The input read file	–read-file	~/workdir/data_artic/basecall_filtered_01.fastq
Number of threads to use	–threads	14
Normalise to max 200fold coverage	–normalise	200
The primer scheme to use	positional (1)	nCoV-2019/V3
The sample name (prefix for output)	positional (2)	barcode_01

Enter the newly created results directory first:

```
cd ~/workdir/results_artic/
```

Then you can run the ARTIC pipeline for one dataset:

```
artic minion --medaka --normalise 200 --threads 14 --scheme-directory ~/artic-
↪ncov2019/primer_schemes --read-file ~/workdir/data_artic/basecall_filtered_01.fastq␣
↪nCoV-2019/V3 barcode_01
```

Perform that step for the first (01) dataset only to save time. Do the other datasets later, when there is time left.

A loop to process all datasets would look like this:

```
for i in {1..5}
do
artic minion --medaka --normalise 200 --threads 14 --scheme-directory ~/artic-
↪ncov2019/primer_schemes --read-file ~/workdir/data_artic/basecall_filtered_0$i.
↪fastq nCoV-2019/V3 barcode_0$i
done
```

When you are done, consensus files have been generated:

```
~/workdir/results_artic/barcode_01.consensus.fasta
```

If you want, you can map the consensus to the Wuhan reference and view the results in GenomeView, or use QUAST, to compare the sequences.

References

ARTIC bioinformatics SOP https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html

1.9.2 Polishing with pilon

Pilon is a software tool which can be used to automatically improve draft assemblies or to find variation among strains, including large event detection. Pilon requires as input a FASTA file of the genome along with one or more BAM files of reads aligned to the input FASTA file. Pilon uses read alignment analysis to identify inconsistencies between the input genome and the evidence in the reads. It then attempts to make improvements to the input genome, including:

- Single base differences
- Small indels
- Larger indel or block substitution events
- Gap filling

- Identification of local misassemblies, including optional opening of new gaps

Pilon then outputs a FASTA file containing an improved representation of the genome from the read data and an optional VCF file detailing variation seen between the read data and the input genome.

To aid manual inspection and improvement by an analyst, Pilon can optionally produce tracks that can be displayed in genome viewers such as IGV and GenomeView, and it reports other events (such as possible large collapsed repeat regions) in its standard output. More information on pilon can be found here: https://github.com/broadinstitute/pilon/wiki

For all 5 barcodes, we have some Illumina data prepared, which you already downloaded to:

```
~/workdir/data_artic/Illumina_<number>
```

We are now going to map the data to the consensus sequence generated by the ARTIC pipeline and generate a new consensus.

Mapping of Illumina reads to assembly

We are mapping the Illumina reads our consensus sequence with BWA. BWA is a software package for mapping low-divergent sequences against a large reference genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate.

Mapping the Illumina data with bwa

First, we need to create an index on the consensus sequences:

```
bwa index ~/workdir/results_artic/barcode_01.consensus.fasta
```

Then, we run the actual mapping with:

```
bwa mem
```

and the following parameters:

Note: You don't need to create a SAM file, when you pipe the results of `bwa mem` directly to `samtools` for SAM to BAM conversion and sorting:

```
<bwa command> | samtools view -b - | samtools sort - > ~/workdir/mappings/Illumina_vs_
↪consensus_01.sorted.bam
```

The complete commandline for bwa mem is:

```
bwa mem -t 14  ~/workdir/results_artic/barcode_01.consensus.fasta  ~/workdir/data_
↪artic/Illumina_01/B0001_S1_L001_R1_001.fastq.gz ~/workdir/data_artic/Illumina_01/
↪B0001_S1_L001_R2_001.fastq.gz > ~/workdir/mappings/Illumina_vs_consensus_01.sam
```

Then convert to BAM (if you didn't created a sorted BAM file directly):

```
samtools view -b ~/workdir/mappings/Illumina_vs_consensus_01.sam | samtools sort - > ~
↪/workdir/mappings/Illumina_vs_consensus_01.sorted.bam
```

And finally, no matter how you created the sorted BAM file, create an index on it:

```
samtools index ~/workdir/mappings/Illumina_vs_consensus_01.sorted.bam
```

Perform that step for the first (01) dataset only to save time. Do the other datasets later, when there is time left.

References

BWA http://bio-bwa.sourceforge.net/

samtools http://www.htslib.org

Mapping of Illumina reads to assembly (2)

This is just a loop-solution to map all samples. You have already mapped one sample, skip this for the moment and come back later, if there is time left.

A loop, to to the indexing of the consensus sequence, mapping and SAM to BAM conversion for you would look like this:

```
for i in {1..5}
do
bwa index ~/workdir/results_artic/barcode_0$i.consensus.fasta
bwa mem -t 14  ~/workdir/results_artic/barcode_0$i.consensus.fasta  ~/workdir/data_
↪artic/Illumina_0$i/B000${i}_S${i}_L001_R1_001.fastq.gz ~/workdir/data_artic/
↪Illumina_0${i}/B000${i}_S${i}_L001_R2_001.fastq.gz | samtools view -b - | samtools␣
↪sort > ~/workdir/mappings/Illumina_vs_consensus_0$i.sorted.bam
samtools index ~/workdir/mappings/Illumina_vs_consensus_0$i.sorted.bam
done
```

We will use the mappings now to call Pilon.

References

BWA http://bio-bwa.sourceforge.net/

samtools http://www.htslib.org

Call Pilon

In the next step, we call pilon with the mappings to polish our consensus sequences.

Check the usage of Pilon:

```
java -jar ~/pilon-1.23.jar --help

Pilon version 1.23 Mon Nov 26 16:04:05 2018 -0500

    Usage: pilon --genome genome.fasta [--frags frags.bam] [--jumps jumps.bam] [--
↪unpaired unpaired.bam]
                 [...other options...]
           pilon --help for option details


         INPUTS:
```

(continues on next page)

(continued from previous page)

```
           --genome genome.fasta
              The input genome we are trying to improve, which must be the reference␣
↪used
              for the bam alignments.  At least one of --frags or --jumps must also␣
↪be given.
           --frags frags.bam
              A bam file consisting of fragment paired-end alignments, aligned to the␣
↪--genome
              argument using bwa or bowtie2.  This argument may be specifed more than␣
↪once.
           --jumps jumps.bam
              A bam file consisting of jump (mate pair) paired-end alignments,␣
↪aligned to the
              --genome argument using bwa or bowtie2.  This argument may be specifed␣
↪more than once.
           --unpaired unpaired.bam
              A bam file consisting of unpaired alignments, aligned to the --genome␣
↪argument
              using bwa or bowtie2.  This argument may be specifed more than once.
           --bam any.bam
              A bam file of unknown type; Pilon will scan it and attempt to classify␣
↪it as one
              of the above bam types.
        OUTPUTS:
           --output prefix
              Prefix for output files
           --outdir directory
              Use this directory for all output files.
           --changes
              If specified, a file listing changes in the <output>.fasta will be␣
↪generated.
           --vcf
              If specified, a vcf file will be generated
           --vcfqe
               If specified, the VCF will contain a QE (quality-weighted evidence)␣
↪field rather
               than the default QP (quality-weighted percentage of evidence) field.
           --tracks
               This options will cause many track files (*.bed, *.wig) suitable for␣
↪viewing in
               a genome browser to be written.
        CONTROL:
           --variant
              Sets up heuristics for variant calling, as opposed to assembly␣
↪improvement;
              equivalent to "--vcf --fix all,breaks".
           --chunksize
              Input FASTA elements larger than this will be processed in smaller␣
↪pieces not to
              exceed this size (default 10000000).
           --diploid
              Sample is from diploid organism; will eventually affect calling of␣
↪heterozygous SNPs
           --fix fixlist
              A comma-separated list of categories of issues to try to fix:
                "snps": try to fix individual base errors;
                "indels": try to fix small indels;
```

(continues on next page)

(continued from previous page)

```
                "gaps": try to fill gaps;
                "local": try to detect and fix local misassemblies;
                "all": all of the above (default);
                "bases": shorthand for "snps" and "indels" (for back compatibility);
                "none": none of the above; new fasta file will not be written.
              The following are experimental fix types:
                "amb": fix ambiguous bases in fasta output (to most likely␣
↪alternative);
                "breaks": allow local reassembly to open new gaps (with "local");
                "circles": try to close circlar elements when used with long␣
↪corrected reads;
                "novel": assemble novel sequence from unaligned non-jump reads.
           --dumpreads
              Dump reads for local re-assemblies.
           --duplicates
              Use reads marked as duplicates in the input BAMs (ignored by default).
           --iupac
              Output IUPAC ambiguous base codes in the output FASTA file when␣
↪appropriate.
           --nonpf
              Use reads which failed sequencer quality filtering (ignored by default).
           --targets targetlist
              Only process the specified target(s).  Targets are comma-separated, and␣
↪each target
              is a fasta element name optionally followed by a base range.
              Example: "scaffold00001,scaffold00002:10000-20000" would result in␣
↪processing all of
              scaffold00001 and coordinates 10000-20000 of scaffold00002.
              If "targetlist" is the name of a file, each line will be treated as a␣
↪target
              specification.
           --threads
              Degree of parallelism to use for certain processing (default 1).␣
↪Experimental.
           --verbose
              More verbose output.
           --debug
              Debugging output (implies verbose).
           --version
              Print version string and exit.
        HEURISTICS:
           --defaultqual qual
              Assumes bases are of this quality if quals are no present in input BAMs␣
↪(default 10).
           --flank nbases
              Controls how much of the well-aligned reads will be used; this many␣
↪bases at each
              end of the good reads will be ignored (default 10).
           --gapmargin
              Closed gaps must be within this number of bases of true size to be␣
↪closed (100000)
           --K
              Kmer size used by internal assembler (default 47).
           --mindepth depth
              Variants (snps and indels) will only be called if there is coverage of␣
↪good pairs
              at this depth or more; if this value is >= 1, it is an absolute depth,␣
↪if it is a
```

(continues on next page)

(continued from previous page)

```
                fraction < 1, then minimum depth is computed by multiplying this value␣
↪by the mean
                coverage for the region, with a minumum value of 5 (default 0.1: min␣
↪depth to call
                is 10% of mean coverage or 5, whichever is greater).
             --mingap
                Minimum size for unclosed gaps (default 10)
             --minmq
                Minimum alignment mapping quality for a read to count in pileups␣
↪(default 0)
             --minqual
                Minimum base quality to consider for pileups (default 0)
             --nostrays
                Skip making a pass through the input BAM files to identify stray pairs,␣
↪that is,
                those pairs in which both reads are aligned but not marked valid␣
↪because they have
                inconsistent orientation or separation. Identifying stray pairs can␣
↪help fill gaps
                and assemble larger insertions, especially of repeat content.  However,␣
↪doing so
                sometimes consumes considerable memory.
```

Start java with 32G memory:

```
java -Xmx32G -jar pilon-1.23.jar
```

Specify a genome sequence, the BAM file with fragment paired end alignments, select the option to generate a file with all changes, and use 14 threads. The output prefix should be:

```
~/workdir/results_artic/barcode_01_pilon
```

As usual, you can get help on the next page.

References

pilon https://github.com/broadinstitute/pilon/wiki

Call Pilon (2)

Start Pilon with:

```
java -Xmx32G -jar pilon-1.23.jar
```

and the following parameters:

What?	parameter	Our value
The consensus sequence	–genome	~/workdir/results_artic/barcode_01.consensus.fasta
The input mapping	–frags	~/workdir/mappings/Illumina_vs_consensus_01.sorted.bam
The output prefix	–output	~/workdir/results_artic/barcode_01_pilon
The number of threads to be used	–threads	14
Generate a file with changes	–changes	

Make sure, to enter the resuls_artic directory first:

```
cd ~/workdir/results_artic/
```

The complete command is:

```
java -Xmx32G -jar ~/pilon-1.23.jar --genome ~/workdir/results_artic/barcode_01.
↪consensus.fasta --changes --frags ~/workdir/mappings/Illumina_vs_consensus_01.
↪sorted.bam --threads 14 --output ~/workdir/results_artic/barcode_01_pilon
```

Perform that step for the first (01) dataset only to save time. Do the other datasets later, when there is time left.

A loop to do that for all samples would look like this:

```
for i in {1..5}
do
java -Xmx32G -jar ~/pilon-1.23.jar --genome ~/workdir/results_artic/barcode_0$i.
↪consensus.fasta --changes --frags ~/workdir/mappings/Illumina_vs_consensus_0$i.
↪sorted.bam --threads 14 --output ~/workdir/results_artic/barcode_0$i_pilon
done
```

Inspect the `barcode_01_pilon.changes` file to see what has been corrected. Usually, you would do several rounds of Pilon until there are no further changes in the changes file. You would need to map the Illumina reads again vs the Pilon-polished consensus sequene, run Pilon, map again, and so on.

We will now just use our first round Pilon polished consensus sequences and put them into nextstrain.

References

pilon https://github.com/broadinstitute/pilon/wiki

References

pilon https://github.com/broadinstitute/pilon

1.9.3 Get the remaining results

If you feel like you want to repeat the previous steps for the remaining datasets and if you are good in time, feel free to do so.

Please rename the result file you have generated to:

```
~/workdir/results_artic/barcode_01.fasta
```

with:

```
mv ~/workdir/results_artic/barcode_01_pilon.fasta  ~/workdir/results_artic/barcode_01.
↪fasta
```

Then download the remaining results (pick the commands accordingly, you don't need barcode_01.fasta, if you succeeded in the previous steps):

```
cd ~/workdir/results_artic/
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪01.fasta
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪02.fasta
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪03.fasta
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪04.fasta
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/barcode_
↪05.fasta
```

Now we can start to use nextstrain.

1.9.4 Nextstrain

Nextstrain is an open-source project to harness the scientific and public health potential of pathogen genome data. It provides a continually-updated view of publicly available data alongside powerful analytic and visualization tools for use by the community. The goal is to aid epidemiological understanding and improve outbreak response.

Nextstrain can also be installed locally and is already installed in your Virtual Machine. We are going to start Nextstrain with an example SARS-Cov2 dataset provided by nextstrain. Then we will include our own data.

Get tutorial dataset

We will download the tutorial dataset for SARS-Cov2 provided by nextstrain. First, enter your workdir:

```
cd ~/workdir/
```

Then clone the repository:

```
git clone https://github.com/nextstrain/ncov.git
```

The data is contained in the directory:

```
ls -l ~/workdir/ncov/data/

total 1724
-rwxrwxr-x 1 ubuntu ubuntu  171914 Nov 19 09:26 example_metadata.tsv
-rwxrwxr-x 1 ubuntu ubuntu 1589835 Nov 19 09:26 example_sequences.fasta.gz
```

There is a fasta file and a metadata file containing information about each sequence.

Extract the data fasta file, located in the repository:

```
gunzip ~/workdir/ncov/data/example_sequences.fasta.gz
```

In the next step, we run the nextstrain basic workflow.

Inspect the data files with less or more:

```
less ~/workdir/ncov/data/example_sequences.fasta
```

and:

```
less ~/workdir/ncov/data/example_metadata.tsv
```

... to get an impression of what data needs to be provided for nextstrain.

References

Nextstrain SARS Cov2 Tutorial https://nextstrain.github.io/ncov/

Start nextstrain

As with the ARTIC pipeline, nextstrain is also installed in a conda environment. Activate it with:

```
conda activate nextstrain
```

Make sure, the conda environment is active during the next steps.

Run the basic nextstrain worflow with:

```
cd ~/workdir/ncov/
snakemake --cores 14 --profile ./my_profiles/getting_started
```

Then start the auspice server:

```
cd ~/workdir/ncov
nextstrain view auspice/
```

And view the results in a browser with:

```
firefox http://127.0.0.1:4000
```

If you are done inspecting the results, terminate the auspice server with `Ctrl+C` .

We will now include our own data and recompute the analysis.

References

Nextstrain SARS Cov2 Tutorial https://nextstrain.github.io/ncov/

Include own data

As we have already seen, the data files are located in:

```
~/workdir/ncov/data/
```

All you need to do, to include your own data, is append the consensus fasta sequences to the fasta file, and add metadata to the metadata file.

Add fasta sequences

Copy the consensus sequences to the data folder:

```
cat ~/workdir/results_artic/barcode_0?.fasta > ~/workdir/ncov/data/consensus_
↪sequences.fasta
```

Then open the sequence file in an editor:

```
gedit ~/workdir/ncov/data/consensus_sequences.fasta
```

and give the following short names to the sequences:

```
Germany/OWL-B0001/2020
Germany/OWL-B0002/2020
Germany/OWL-B0003/2020
Germany/OWL-B0004/2020
Germany/OWL-B0005/2020
```

Attention: Please use exactly the names above, since these are the names, we provide in the metadata file below.

Then append the fasta file to the example sequences with (maybe inspect it with `less` or `more` before):

```
cat  ~/workdir/ncov/data/consensus_sequences.fasta >> ~/workdir/ncov/data/example_
↪sequences.fasta
```

Add metadata

Download the metadata for our samples:

```
cd ~/workdir/ncov/
wget https://openstack.cebitec.uni-bielefeld.de:8080/swift/v1/coursedata2020/metadata.
↪tsv
```

Append it to the example metadata file:

```
cat ~/workdir/ncov/metadata.tsv >> ~/workdir/ncov/data/example_metadata.tsv
```

Rerun analysis

Then make sure, the nextstrain environment is active. If not:

```
conda activate nextstrain
```

Run the basic nextstrain workflow with:

```
cd ~/workdir/ncov/
snakemake --cores 14 --profile ./my_profiles/getting_started
```

Then start the auspice server:

```
cd ~/workdir/ncov
nextstrain view auspice/
```

View the results in a browser with:

```
firefox http://127.0.0.1:4000
```

That's it for the short nextstrain tutorial, you can check out the official nextstrain site for many more things you can do with nextstrain, which is way beyond the scope of this workshop.

References

Nextstrain SARS Cov2 Tutorial https://nextstrain.github.io/ncov/

References

Nextstrain SARS Cov2 Tutorial https://nextstrain.github.io/ncov/

1.10 Outlook

If you are further interested in working with Nextstrain, you can check out the SARS-Cov2 Tutorial and go on with some steps, for example customizing your analysis or visualisation:

https://nextstrain.github.io/ncov/

And if you are totally enthusiastic about Nanopore data, you can do the course of last year, where we analysed two bacterial genomes. The course is still available online and you could follow the instructions. In some cases, the mapping tools used are not installed in your VM, then just use Minimap2 or BWA (depending on the data) and use GenomeView instead of IGV. We can't guarantee that everything works exactly as last year, since some tool versions also changed. You can find the tutorial here:

https://denbi-nanopore-training-course.readthedocs.io/en/latest/