Metagenomic methods for microbial ecologists Documentation

Release 1.0

Mattias de Hollander

September 18, 2014

Contents

1 Day 1: Introduction **3**
- 1.1 Preprocessing & quality filtering using a Snakemake workflow . . . 3
- 1.2 Introduction to Snakemake . . . 3
- 1.3 Launch Snakemake . . . 3
- 1.4 Create a rule to unzip FASTQ files . . . 4
- 1.5 Creating quality plots using FASTQC . . . 5
- 1.6 Trim low quality reads and remove adapters . . . 6
- 1.7 Create quality plots of the trimmed data . . . 6
- 1.8 Create quality plots using R . . . 6
- 1.9 Merge paired-end Illumina data . . . 7

2 Day 4: Metagenomic assembly and binning **11**
- 2.1 Installing perl-doc (to view some help files for ESOM) . . . 11
- 2.2 Qality checking your data . . . 11
- 2.3 running cutadapt . . . 12
- 2.4 Running mira . . . 13
- 2.5 Binning . . . 14
- 2.6 ESOM . . . 15
- 2.7 crAss . . . 15
- 2.8 Differential coverage: manual . . . 15
- 2.9 groopM . . . 16

Contents:

CHAPTER 1

Day 1: Introduction

Today we will give an introduction to sequencing techniques and data processing with linux.

1.1 Preprocessing & quality filtering using a Snakemake workflow

In this part the following topics will be covered

- introduction to Snakemake
- unzipping files
- making FASTQC plots
- trimming reads with low quality and adapter removal
- combining all steps into a single workflow

1.2 Introduction to Snakemake

https://speakerdeck.com/johanneskoester/workflow-management-with-snakemake

- Documentation for snakemake can be found here: https://bitbucket.org/johanneskoester/snakemake/wiki/Documentation
- Use this as a reference while following the step-by-step guide below

1.3 Launch Snakemake

- Boot up the Ubuntu live USB-stick
- Open a terminal window using the icon on the left on the screen or by typing: `ctrl + alt + T`
- Start the Snakemake environment:

  ```
  source /usr/local/bin/virtualenvwrapper.sh && workon snakemake
  ```

- Make sure the snakemake executable is available:

  ```
  snakemake -h
  ```

Note: **Question**: Which version of Snakemake is installed?

1.4 Create a rule to unzip FASTQ files

- Create a directory for this part for the workshop:

```
mkdir qc_snakemake
cd qc_snakemake
```

In Snakemake, workflows are specified as Snakefiles. Inspired by GNU Make, a Snakefile contains rules, that denote how to create output files from input files. Dependencies between rules are handled implicitly, by matching filenames of input files against output files. Thereby wildcards can be used to write general rules.

- Create the main workflow file using your favorite text editor. For a graphical editor Gedit is recommended and for a command-line editor Nano is a good choice.:

```
gedit Snakefile &
```

- Add the following lines:

```
configfile: "config.json"

rule all:
    input:
        expand("bunzip/{sample}.fastq",sample=config['data'])

rule bunzip:
    input:
        lambda wildcards: expand("{basedir}{data}",basedir=config["basedir"],data=config["data"]
    output:
        "bunzip/{data}.fastq"
    log:
        "bunzip/{data}.log"
    threads: 1
    shell: "bunzip2 -d -c {input} > {output}"
```

- Save the file. If you are using Gedit use the File menu or type `ctrl + s` or in Nano type `ctrl + x` and `y`

In the above `Snakefile` first the configuration file is loaded, next a target rule is defined that will trigger the creation of the required output. Finally a rule is made to unzip FASTQ files compressed in bzip2 format.

Since version 3.1, Snakemake directly supports the configuration of your workflow. A configuration is provided as a JSON file. The JSON file can be used to define a dictionary of configuration parameters and their values. In the workflow, the configuration is accessible via the global variable `config`.

- Create a config file:

```
gedit config.json
```

- Add the following configuration:

```
{
    "basedir": "/home/nmp/Documents/DeHollander/",
    "samples": {
        "Sample1": ["Sample1"],
        "Sample2": ["Sample2"],
        "Sample3": ["Sample3"]
    },
    "data": {
        "Sample1": ["sample1.fastq.bz2"],
        "Sample2": ["sample2.fastq.bz2"],
        "Sample3": ["sample3.fastq.bz2"]
```

```
    },
    "adapters": "/home/nmp/Documents/DeHollander/contaminant_list.txt",
    "adapters_fasta": "/home/nmp/Documents/DeHollander/illumina_truseq_adapters.fa"
}
```

- Now run snakemake. It will unpack the input files using the `bunzip2` command:

```
snakemake
```

Note:

Questions:

- Can you explain what happened? Which files are used for input? Which output files are created?
- Open one of the fastq files either using the File Browser or using `less` on the command line

---

1.5 Creating quality plots using FASTQC

- Add the following text rule to the Snakefile after the bunzip rule:

```
rule fastqc_raw:
    input: fastq="bunzip/{sample}.fastq"
    output: "fastqc_raw/{sample}_fastqc/"
    params: dir="fastqc_raw", adapters=config['adapters']
    log: "fastqc_raw.log"
    threads: 1
    shell: "fastqc -q -t {threads} --contaminants {params.adapters} --outdir {params.dir} {input
```

- Add final fastqc outputs to the target rule `all`. Is should now look like this (note that a comma is added at the end of the bunzip line):

```
rule all:
    input:
        expand("bunzip/{sample}.fastq",sample=config['data']),
        expand("fastqc_raw/{sample}_fastqc/",sample=config['data'])
```

- Make the plots by running Snakemake again:

```
snakemake
```

- Have a look at the results using a browser:

```
firefox fastqc_raw/Sample*_fastqc/fastqc_report.html
```

Note:

Questions:

- What are the difference between the samples?
- Is there any difference in quality?
- Can you find adapter contamination?

1.6 Trim low quality reads and remove adapters

- Create a rule for the fastq-mcf program:

```
rule trim:
    input: fastq="bunzip/{sample}.fastq"
    output: fastq="fastq-mcf/{sample}_trimmed.fastq"
    params: dir="fastq-mcf", adapters=config["adapters_fasta"]
    log: "{sample}_fastq-mcf.log"
    threads: 1
    # -C: number of subsamples used for determining adapter parameters
    # http://scotthandley.wordpress.com/2013/09/25/adapter-removal-and-adaptive-quality-trimming
    shell: "fastq-mcf {params.adapters} {input.fastq} -o {output.fastq} -C 99999 -P 33 -w 4 -q 2
```

- Add it to the Snakefile after the fastqc-raw rule
- Run snakemake now with a parameter showing the command that is going to be performed:

```
snakemake -p
```

Note:

Exercise:

- Have a look at the fastq-mcf parameters. What is the minimum quality.
- What do the other parameters mean?

1.7 Create quality plots of the trimmed data

Note:

Exercise:

- Create a rule for creating plots with FASTQC on the trimmed data.
- Use the fastqc_rule as as a template and adjust the input and output parameters.
- Add the fastqc_trim output to the `rule all:` by adding another expand() line with the output of the fastqc_trim rule as input.
- Run snakemake
- Open the output files in Firefox and compare them to the raw data. What is the effect of trimming the reads?

1.8 Create quality plots using R

Is it possible to run R code directly from a Snakefile. To make similar quality plots as FASTQC in R we are going to make use of the 'qrqc' package: http://www.bioconductor.org/packages/release/bioc/html/qrqc.html

- First we need to install some required software on the USB. Copy paste the commands below to perform this installation:

```
wget https://bitbucket.org/johanneskoester/snakemake/raw/ef669ba49226368fc71eb1975a89168dcb87d59
mv utils.py ~/.virtualenvs/snakemake/lib/python3.3/site-packages/snakemake/
sudo apt-get -y install python3-dev
pip install rpy2
```

- Create a rule that reads in the untrimmed and trimmed data and saves a quality plot to disk:

```
from snakemake.utils import R

rule qc_plot:
    input: untrimmed="bunzip/{sample}.fastq", trimmed="fastq-mcf/{sample}_trimmed.fastq"
    output: "qrqc/{sample}_qualplot.png"
    run:
        R("""
        library("qrqc")

        s1 <- readSeqFile("{input.untrimmed}")
        s2 <- readSeqFile("{input.trimmed}")
        q <- qualPlot(list(untrimmed = s1, trimmed = s2))
        ggsave("{output}", q)
        """)
```

- Add this rule again to the Snakefile.
- Adjust the `rule all:` to incude the output of the qc_plot rule.

Note: **Exercise**: Extend above rule to the other functions of the qrqc package. Have a look at the documentation of the qrqc package to see what is possible. For example create a base frequency plot with the basePlot command.

1.9 Merge paired-end Illumina data

Sequence that are produced by the Illumina HiSeq or MiSeq machines come in pairs of fastq files with each pair of reads. Files containing R1 in the filename are in the forward direction and with R2 in the reverse. The two read pairs can be merged into a single, longer sequence. Downstream analysis can benefit of the longer read length. Several tools excist to merge paired-end read, and in this section we will make use of FLASH (http://ccb.jhu.edu/software/FLASH/).

- First we need to setup a new directory to store the results:

```
mkdir /home/nmp/mergepairs && cd /home/nmp/mergepairs
```

- Create a new config.json file with this content:

```
{
    "basedir": "/home/nmp/Documents/DeHollander/",
    "samples": {
        "Sample4": ["Sample4"],
        "Sample5": ["Sample5"]
    },
    "data": {
        "Sample4": {"forward": "sample4_R1.fastq", "reverse": "sample4_R2.fastq"},
        "Sample5": {"forward": "sample5_1.fq", "reverse": "sample5_2.fq"}
    },
    "adapters_fasta": "/home/nmp/Documents/DeHollander/illumina_truseq_adapters.fa"
}
```

- The first step again is to trim the data. Since in the previous section we have worked with single end data, we have to adjust the fastq-mcf rule:

```
rule fastqmcf_paired:
    input:
        forward = lambda wildcards: config["basedir"] + config["data"][wildcards.data]['forward'
        reverse = lambda wildcards: config["basedir"] + config["data"][wildcards.data]['reverse'
    output:
        forward="fastq-mcf/{data}_R1_trimmed.fastq",
        reverse="fastq-mcf/{data}_R2_trimmed.fastq"
    params: dir="fastq-mcf", adapters=config["adapters_fasta"]
    log: "{data}_fastq-mcf.log"
    threads: 1
    # -C: number of subsamples used for determining adapter parameters
    # http://scotthandley.wordpress.com/2013/09/25/adapter-removal-and-adaptive-quality-trimming
    shell: "fastq-mcf {params.adapters} {input} -o {output.forward} -o {output.reverse} -C 99999
```

- We need to get a list of samples from the config file. Add this line to the Snakefile after the config.json file is loaded:

```
DATA_TO_SAMPLE = {
    data: sample for sample, datasets in config["samples"].items()
    for data in datasets}
```

- Now lets run Snakemake to create a trimmed file of the forward read from Sample4:

```
snakemake fastq-mcf/Sample4_R1_trimmed.fastq
```

One of the strengths of Snakemake is that you can run it on multiple input files with a range of parameters automatically. We are going to run the FLASH program on Sample4 and Sample5 with a range of mismatch parameters. Therefore we need to create a MISMATCHES variable and a 'final' rule that will trigger the creation of all needed files.

- Add these lines directly after the `DATA_TO_SAMPLE` variable:

```
MISMATCHES = "0 0.1 0.5".split(' ')

rule all:
    input:
        expand("flash/{data}_trimmed_{mismatch}.extendedFrags.fastq".split(),data=config["sample
```

- Now add a rule to run FLASH on a range of mismatch values:

```
rule flash_trimmed:
    input:
        forward="fastq-mcf/{data}_R1_trimmed.fastq",
        reverse="fastq-mcf/{data}_R2_trimmed.fastq"
    output:
        "flash/{data}_trimmed_{mismatch}.extendedFrags.fastq"
    params:
        sample=lambda wildcards: DATA_TO_SAMPLE[wildcards.data],
        prefix="{data}_trimmed_{mismatch}", dir="flash",minlength="5",maxlength="400",mismatch="
    log:
        "{data}_trimmed_{mismatch}_mergepairs.log"
    shell:
        "flash -m {params.minlength} -M {params.maxlength} {input} -x {params.mismatch} -o {para
```

- To visualize the steps Snakemake is going to perform on the different datasets, we can create a dependency graph with the following command:

```
$ snakemake --dag | dot | display
```

- Finally, let's run Snakemake to create all of the files seen in the previous picture:

```
snakemake
```

Note:

Questions:

- Have a look at the fastq-mcf log file `fastq-mcf/Sample4_fastq-mcf.log`. Do you see any adapter contamination? Do the same for Sample5.
- How many reads are merged? Compare the FLASH log file for Sample4 and Sample5: `flash/Sample4_trimmed_0_mergepairs.log`
- What is the effect of the different mismatch values?

The main purpose of merging read pairs is to get longer reads. Using the qrqc package in `R` we can compare the length of the sequences before and after merging.

- Create the following `qrqc_seqlen` rule at the end of Snakefile:

```
from snakemake.utils import R

rule qrqc_seqlen:
    input: untrimmed="fastq-mcf/{data}_R1_trimmed.fastq", trimmed="flash/{data}_trimmed_0.extend
    output: "qrqc/{data}_lengthplot.png"
    run:
        R("""
        library("qrqc")

        s1 <- readSeqFile("{input.untrimmed}")
        s2 <- readSeqFile("{input.trimmed}")
        l <- seqlenPlot(list(untrimmed = s1, trimmed = s2))
        ggsave("{output}", l)
        """)
```

- Create the plot for Sample4 and have a look at it:

```
snakemake qrqc/Sample4_lengthplot.png
display qrqc/Sample4_lengthplot.png
```

Note:

Optional exercise:

- Make a rule to merge data with FLASH using the untrimmed files
- There are alternative programs to perform the trimming and merging. Create rules for Alientrimmer to trim the data and Usearch to merge data `usearch -fastq_mergepairs`. Compare this with the output of Fastq-mcf and FLASH.

CHAPTER 2

Day 4: Metagenomic assembly and binning

Note: First a little disclaimer:

In any bioinformatics step, there is not one 'best' tool for the job. Depending on the data you have, you might want to use different programs than we demonstrate here. The programs used here are chosen because we have (some) experience running them and they work with the data we have (400bp single end Ion Torrent reads). We have tested most of the the workflow we demonstrate today, but there will probably be things that don't work. if you run into problems, or have questions, ask!

In any assembly workflow, metagenomic or otherwise, it is important that you remove bad quality and short reads from the data. To assess the quality of the data we use the program 'FastQC'. For the trimming, we use 'cutadapt'.

But first, installing and downloading some extra stuff apparently we have missed a few things on the usb stick

2.1 Installing perl-doc (to view some help files for ESOM)

- Type this in a terminal window (type password Wag2014 when prompted):

```
sudo apt-get install perl-doc
```

- re-installing cutadapt:

```
cd
cd Programs/
rm -r cutadapt-1.4.1
cd
sudo pip install cutadapt
```

- installing samtools:

```
sudo apt-get install samtools
```

2.2 Qality checking your data

- Go to the directory where this mornings files are located:

```
cd ~/Documents/Speth-Dutilh/

# get organized!
```

```
mkdir crAss_data
mv F* crAss_data

mkdir wwtp_data
mv course_NIOO_data* ctab_* kit_* wwtp_data

ls
cd wwtp_data
```

- run fastqc on 4 datasets, without any arguments:

```
fastqc kit_* ctab_*
```

- inspect the data:

```
firefox *_fastqc/fastqc_report.html &
```

Note: this will open 4 tabs on firefox, each tab contains the quality report of one of the datasets the '&' symbol at the end will run firefox in the background, so that we can continue below after pressing enter check the failures and warnings after running cutadapt, and rerunning fastQC, we'll discuss the results briefly

---

2.3 running cutadapt

- get organized again!:

```
mkdir raw_data
mv ctab_* kit_* raw_data
mkdir trimmed_data
cd trimmed_data
```

- briefly look at the options, we'll only use length and quality trimming (press enter):

```
cutadapt -h
```

- run cutadapt, -m specifies minimum length, -q defines minimum quality of sequence ends -o states the name of the output file:

```
cutadapt -m 100 -q 24 -o ctab_total_trimmed.fastq ../raw_data/ctab_total_raw.fastq
cutadapt -m 100 -q 24 -o ctab_wash_trimmed.fastq ../raw_data/ctab_wash_raw.fastq
cutadapt -m 100 -q 24 -o kit_total_trimmed.fastq ../raw_data/kit_total_raw.fastq
cutadapt -m 100 -q 24 -o kit_wash_trimmed.fastq ../raw_data/kit_wash_raw.fastq
```

- run fastqc on the trimmed data:

```
fastqc kit_* ctab_*
```

- inspect the data:

```
firefox *_fastqc/fastqc_report.html &
```

Note: not all data will show passed, but still the graphs will give information on the effect of the trimming. we will discuss the data before Bas will start with his assembly lecture

---

Now that we have trimmed reads, we can prepare for the assembly.

2.4 Running mira

- get organized yet again! (although the following steps are not strictly necessary, it is good practice to keep your data organized.):

```
cd ~/Documents/Speth-Dutilh/wwtp_data
mkdir mira
mkdir mira/data
mkdir mira/assembly
cp trimmed_data/*.fastq mira/data
```

- go to assembly folder and create manifest file:

```
cd mira/assembly
```

- open a text editor:

```
nano
```

- write (or copy/paste) the following lines:

```
project = NIOO_metagenomics_course
job = est,denovo,accurate

readgroup = ctab_total
data = ../data/ctab_total_trimmed.fastq
technology = iontor

readgroup = ctab_wash
data = ../data/ctab_wash_trimmed.fastq
technology = iontor

readgroup = kit_total
data = ../data/kit_total_trimmed.fastq
technology = iontor

readgroup = kit_wash
data = ../data/kit_wash_trimmed.fastq
technology = iontor
```

Note: project (name) and job (parameters) give information on the total assembly readgroup allows us to co-assemble different datasets. This enables for instance hybrid assemblies of Pacbio and illumina data is the readfile (relative to current dir), and technology the platform used for sequencing this is, by far, not an exhaustive list of options

save as manifest.conf to save, press 'control-o', and choose the name to exit nano, press 'control-x'

- run mira:

```
mira manifest.conf >& log_assembly.txt
```

Note: ################## LUNCHTIME! (while Mira runs) ########################## ####### MAKE SURE THAT MIRA IS RUNNING BEFORE YOU GO FOR LUNCH ########### ########### If you don't know how, ask us before you leave ###############

- inspect data:

```
cd ~/Documents/Speth-Dutilh/wwtp_data/mira/assembly/NIOO_metagenomics_course_assembly/NIOO_metag
ls -l
```

Note: All WARNINGS files should be empty (size 0)

- check general assembly stats:

```
more NIOO_metagenomics_course_info_assembly.txt
```

- We'll use the contigstats file to visualize clusters later:

```
head metagenomics_course_info_contigstats.txt

cd ../NIOO_metagenomics_course_d_results/
ls -l
```

Note: The contigs are in NIOO_metagenomics_course_out.unpadded.fasta

2.5 Binning

We'll look into several binning methods.

- GC content and coverage:

```
cd ~/Documents/Speth-Dutilh/wwtp_data/mira/
mkdir binning
cp assembly/NIOO_metagenomics_course_assembly/NIOO_metagenomics_course_d_results/NIOO_metagenomi
cp assembly/NIOO_metagenomics_course_assembly/NIOO_metagenomics_course_d_info/NIOO_metagenomics_
```

- open Rstudio:

```
rstudio &
```

- type the following in Rstudio:

```
install.packages("ggplot2")
library(ggplot2)

setwd("~/Documents/Speth-Dutilh/wwtp_data/mira/binning")

contigstats <- read.delim("stats.txt")
names(contigstats)
plot <- ggplot(contigstats, aes(x=GC., y=av.cov, size=length)) + geom_point()
plot
```

look at those beautiful clusters - let's select one:

```
bin1 <- subset(contigstats, GC. > 40 & av.cov > 12.5)
bin1_plot <- ggplot(bin1, aes(x=GC., y=av.cov, size=length)) + geom_point()
bin1_plot

sum(bin1$length)

write.table(bin1, file="bin1_stats.txt",sep="\t", row.names=FALSE, quote=FALSE)
```

- back to the terminal and type:

```
ls -l
head bin1_stats.txt
```

2.6 ESOM

- change directory to where the binning files are:

```
cd ~/Documents/Speth-Dutilh/wwtp_data/mira/binning
```

- run the esomwrapper script:

```
perl ~/Programs/Binning-master/esomWrapper.pl -p . -e fa -min 1000 -scripts ~/Programs/Binning-m
```

Note: as indicated by the name, this is a wrapper for several other scripts it calculates the tetranucleotide frequencies of the contigs, and prepares the files for ESOM

- look at the log file and check the rows/columns of neurons for the map:

```
more ESOM/esom.log
```

- open the esom software:

```
esomana
```

- we'll take it interactively from here:

```
info on this can be found in the ESOM README "~/Programs/Binning-master/README.md"
```

perl scripts to get from class files to binned contigs come with the ESOM installation

2.7 crAss

- change directory to where the binning files are:

```
cd ~/Documents/Speth-Dutilh/wwtp_data/mira/binning
```

- get organized!:

```
mkdir crAss
cp ../mira/assembly/NIOO_metagenomics_course_assembly/NIOO_metagenomics_course_d_results/NIOO_me
cp ../mira/data/kit_* ../mira/data/ctab_* crAss
```

- convert files:

```
cd crAss
miraconvert -f caf -t ace NIOO_metagenomics_course_out.caf assembly.ace
cat ctab_total_trimmed.fastq | perl -e '$i=0;while(<>){if(/^\@/&&$i==0){s/^\@/\>/;print;}elsif($
cat ctab_wash_trimmed.fastq | perl -e '$i=0;while(<>){if(/^\@/&&$i==0){s/^\@/\>/;print;}elsif($i
cat kit_total_trimmed.fastq | perl -e '$i=0;while(<>){if(/^\@/&&$i==0){s/^\@/\>/;print;}elsif($i
cat kit_wash_trimmed.fastq | perl -e '$i=0;while(<>){if(/^\@/&&$i==0){s/^\@/\>/;print;}elsif($i=

grep -E '^CO|AF' assembly.ace > small_assembly.ace
```

2.8 Differential coverage: manual

- align reads using bowtie 2:

```
mkdir aligned_reads

# build index from contigs
bowtie2-build contigs.fa indexed_contigs
mv indexed_contigs* aligned_reads

cd aligned_reads
```

- repeat the 4 commands below for all four datasets:

```
bowtie2 indexed_contigs -U ../../data/ctab_total_trimmed.fastq -S ctab_total.sam
samtools view -bS ctab_total.sam > ctab_total.bam
samtools sort ctab_total.bam ctab_total.sorted.bam
samtools depth ctab_total.sorted.bam > ctab_total_depth.txt
```

Note: ####### here we'll need a little workaround...######### # we'll download a perlscript to calculate the read depth from or "*_depth.txt" files # After running this perl script we'll look at the data in Rstudio # the script is here: # https://raw.githubusercontent.com/MadsAlbertsen/multi-metagenome/master/misc.scripts/calc.coverage.in.bam.depth.pl

---

2.9 groopM

groopm is an automated binning tool, using differential coverage and kmer content it requires contigs and sorted/indexed BAM files we have sorted our BAM files in the previous steps and will index them now and then follow the steps at: http://minillinim.github.io/GroopM/

Boot the USB Flash Drive in VirtualBox

Sometimes with modern computers the USB drive provided with he course cannot boot. Here, we show you how we booted from a USB Flash Drive in VirtualBox. This process will allow you to run your portable Linux from the USB Flash Drive while still running from Windows. By default VirtualBox does not support USB Boot. However this is easily attainable by mapping a virtual machine to the USB Drive.

what you need * Download Virtual box from the VirtualBox Website: http://www.virtualbox.org and install the software in the default path * The AllBio portable Linux USB stick with Ubuntu 14.04 inserted in one of your USB ports while running Windows * Your Windows administrator password (Vista/Win7) 8.1 should do fine * The root password of your AllBio portable Linux ('g') * The password of your AllBio portable Linux main user nmg ('Wag2014')

Locate your USB drive number

- Click Start > Run Type diskmgmt.msc and click OK (In Vista/Win7 use the Start > Search Box)
- Locate your USB Disk #