Table of contents

An example notebook

If you want to see an example for documented code, check out an example for how I might have recorded how I analysed some sequence data myself.

The link above leads you to an example for:

- How could I use github to make my code available to others
- How could a "code book" look like? The example you see in the folder is provided a Quarto markdown file here and for convenience the report was also rendered as a html to make it easier to read for potential collaborators. To view the HTML report, you need to download it first.

The example is based on the data you analyse throughout this tutorial, so the actual code might only make sense after you have finished the tutorial. Feel free to revisit this page after you have written some code yourself.

Please note that this is just an example to get you started and such a report might look different depending on your needs.

Below you find some general hints for what to put into different sections that you can see in the qmd file.

YAML headers

```yaml
---
title: "My report  for analysis X (project title)"

author: Nina Dombrowski
date: 12/18/2023

format:
  html:
    embed-resources: true

toc: true
toc-depth: 2

execute:
  eval: false

engine: knitr
---
```

The text between the `---` is what is called a YAML header an
completely optional but useful if you document code in Rma
or Quarto.

It can be used to specify:
- The characteristics of your document, i.e. title, authors, dat
creation
- Arguments that control how you render your document, fo
example in this example to HTML
- Other parameters for your report. For example, I do not wa
code cells to executed (eval: false)

Setting up your working directory

Setup working directory

```bash
#path to working directory
cd /path_to_folder_you_analyse_the_data

#activate conda environment
mamba activate seqkit_2.6.1

#custom variables
download_link="https://github.com/ndombrowski/cli_workshop/raw/main/data/seq_project.tar.g
```

In this first section I add everything that a user that wants to use my workflow has to change and I try to standardize the code below, so that another user could just run this as is without having to edit anything. Typically things to add are:

- From where to start the analyses
- Custom paths, to for example for databases that need to be downloaded from elsewhere
- Custom variables: In the example above I store the link to the data in a variable called `download_link`, I then use the variable in the code below to download the data. By doing it this way I have one location in the code another person needs to change the code when for example the path to the data changes. The code below stays the

same. When writing code try to think ahead and minimize the number of instances where things need to be changed if for example the location to your data changes.
- ...

I tend to NOT add the information how I use `scp` to log into an HPC to keep my user name and login information private.

Sanity checks

Perform sanity checks

```
#we work with 8 files
wc -l samples.txt

#we have as many R1 as we have R2 files
ls data/seq_project/*/*R1* | wc -l
ls data/seq_project/*/*R2* | wc -l

#the sequence header looks as expected
zcat data/seq_project/*/*gz | head

#R1 and R2 reads have the same number of reads
for i in `cat samples.txt`; do
    echo "$i: $(( $(zcat data/seq_project/*/$i | wc -l) /4 ))"
done > read_counts_per_sample.txt
```

When I first start working with new data, I try to add as many sanity checks as possible to ensure that my data looks good. That way I avoid that I don't notice an issue and run into trouble further down the line. I at the same time understand my data more and learn with how many samples and how much data I am working with.

I also add such sanity checks whenever I modify my data. For example, when I merge individual files into a large file I might count the lines for the individual files and the combined file simply to ensure that I used my wildcards correctly.

Remember: The computer is only as smart as the person using it and will blindly run your commands. Because of that the computer can do unexpected things and you need to account for that.

Using markdown to make notes

```
mkdir -p results/fastqc

#Submitted batch job 8100
sbatch scripts/run_fastqc.sh

ll results/fastqc/*
```

Comments on the visual inspection of the HTML reports generated by FastQC:

- The sequence quality looks good, most reads have a phred score between 28-40 when looking at the per base quality
- The reads are typically 250 bp long (as expected)
- In the over-represented sequences no hits to Illumina adapters where found

In the example above I use markdown to not only document my code but also add some comments whenever I might need it. For example here, I added some notes about what the results from FastQC actually told me.

This kind of documentation can be useful for:

- Justifying decisions further down the line. In this example, I might decide on how to clean the sequencing data. For example, if I would have found a lot of low quality reads or the adapter still being part of the sequence then I would have specifically cleaned my sequences to deal with that
- Future you. If you read the report a month, or a year, later you have the key information in your report and don't have to open the HTML reports again.

Documenting external scripts

Run FastQC

```
mkdir -p results/fastqc

#Submitted batch job 8100
sbatch scripts/run_fastqc.sh

ll results/fastqc/*
```

This part might make more sense after you have worked through the part of the tutorial about using an HPC. But what you see here is how I have written down code that simply says that I submitted a script to a HPC but it does not actually say how I ran the FastQC software. The actual code is "hidden" inside of the run_fastqc.sh script. This also means that a person reading your workflow does not have the code right away. You can deal with this in two ways.

1. Instead of the sbatch command, you can add the actual line of code that was run on the compute node.
2. When publishing your code with your manuscript, add the whole scripts folder to where you publish the main code, i.e. on github or zenodo

I tend to prefer number 2 because I like to record the code in my notebooks exactly as I ran them but you can do it differently as long as all the code you ran is recorded and accessible to others once you publish your data.