An Introduction to HPC-conformant Scientific Workflows

Workflow Course - Edition 3

The HPC-Group

September 2023

Honour where Honour is due

- Johannes Köster
- Lukas Hellmann
- Christian Meesters

- no name yet to fill the list

Outline

What is this about?

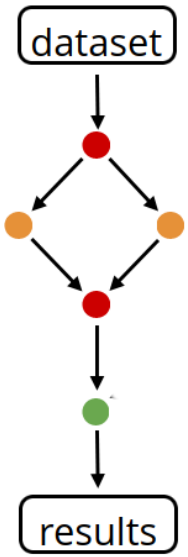
Questions

- I can code everything! Can I?
- What is the benefit of a workflow system?
- What distinguishes a workflow system from a “pipeline”?

Objectives

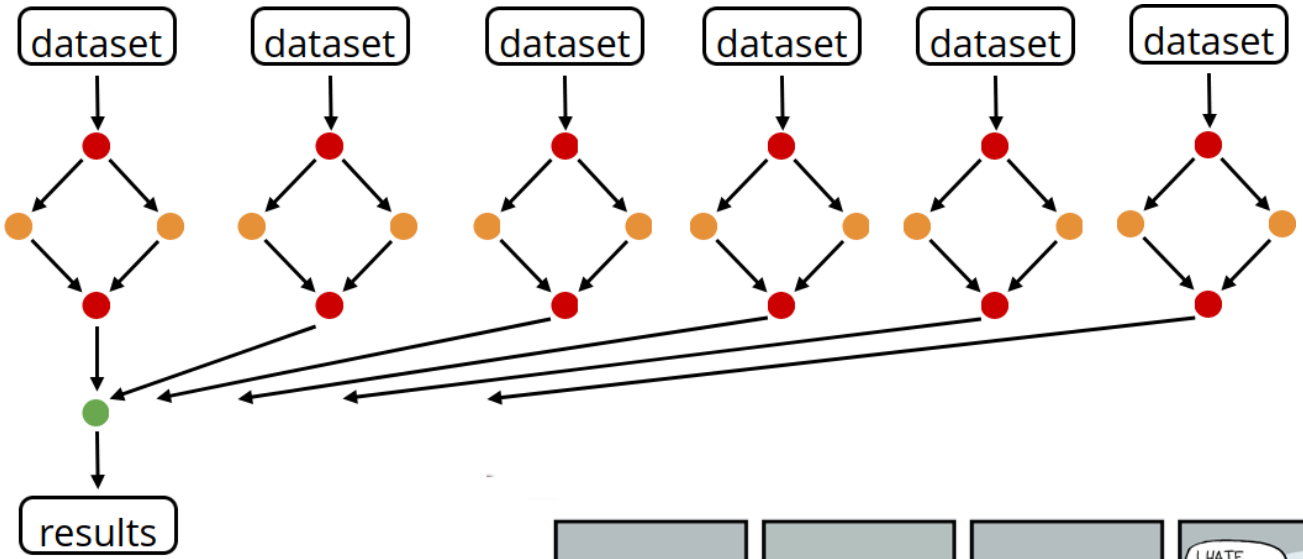
1. Introducing workflow engines (particularly `Snakemake`)!

Data Analysis



Idea from the official `Snakemake` course (with permission), image from PhD comics.

Data Analysis



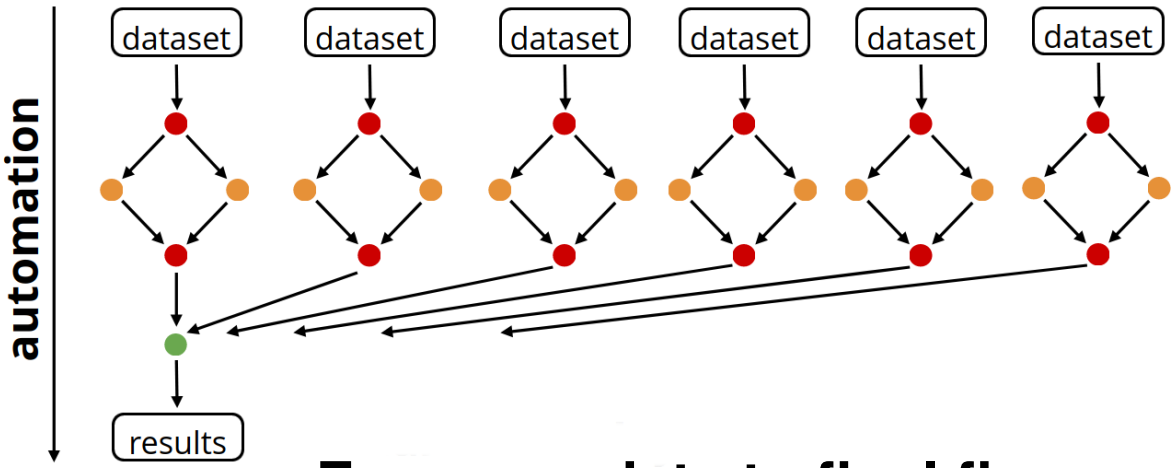
Idea from the official `Snakemake` course (with permission), image from PhD comics.

Goals of Reproducibility

1. Dispel Doubts
2. Facilitate Further Experimentation

Idea from DFN slides.

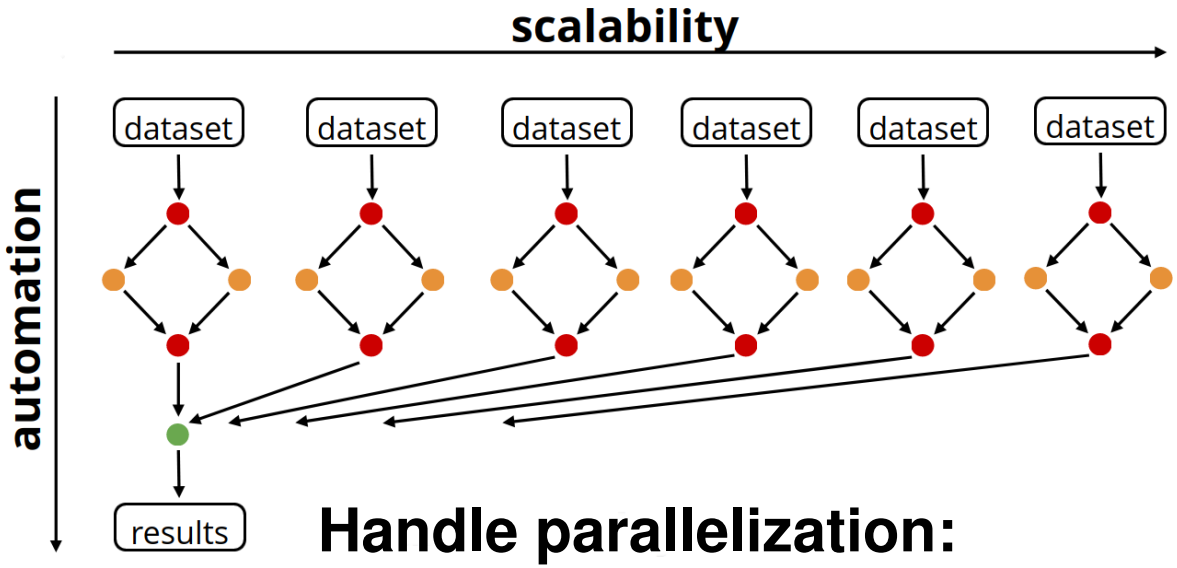
Reproducible Data Analysis



From raw data to final figures:

- **document** parameters, tools, versions
- **execute** without manual intervention

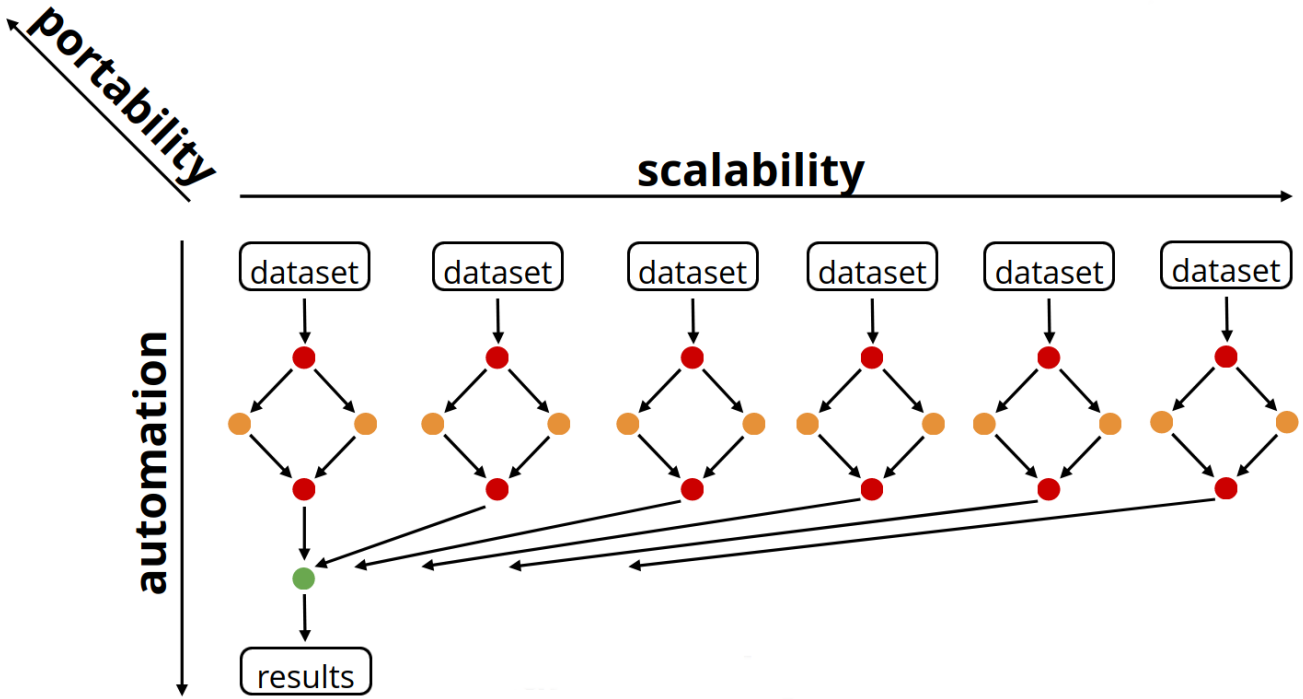
Reproducible Data Analysis



Handle parallelization:

- execute for tens of thousands of datasets
- efficiently use any computing platform

Reproducible Data Analysis



Handle deployment:
be able to easily execute analyses on a different system/platform/infrastructure

Beyond Reproducibility

Reproducibility

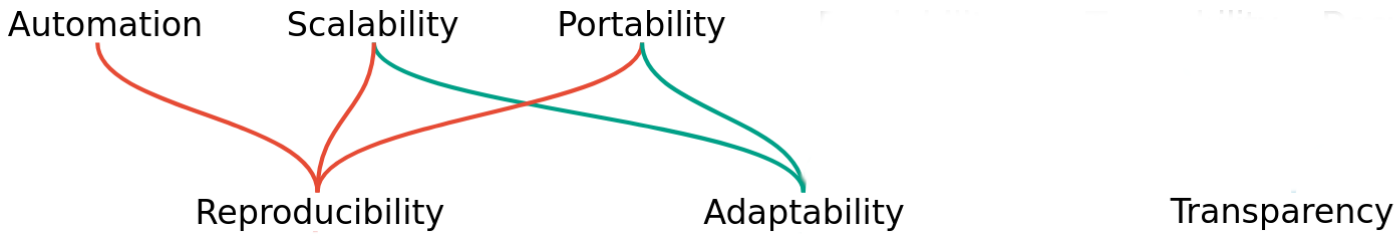
Beyond Reproducibility

Reproducibility

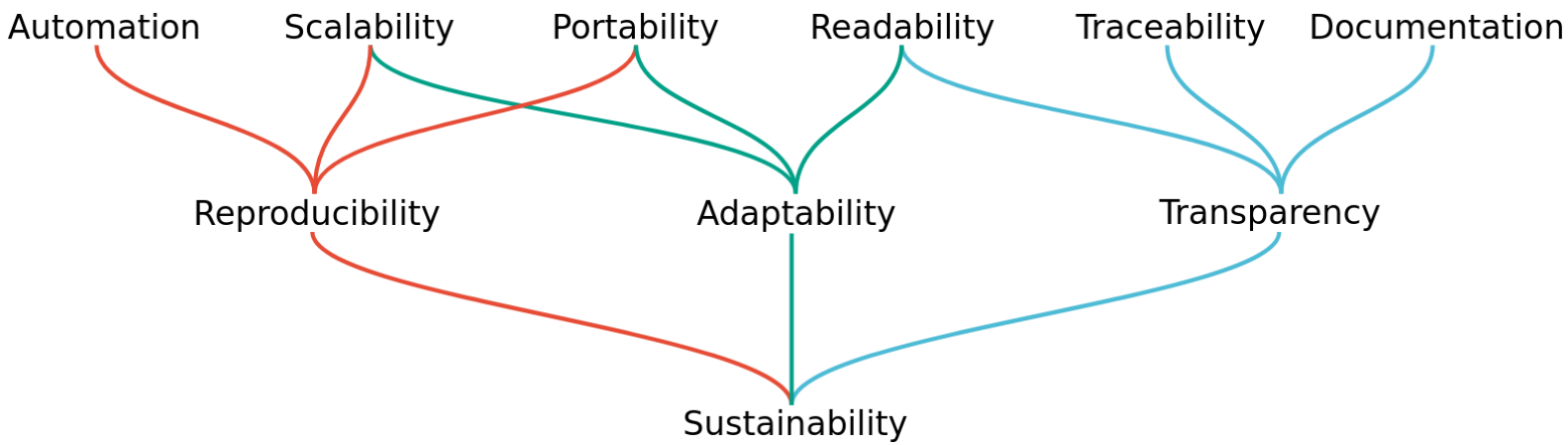
Adaptability

Transparency

Beyond Reproducibility



Beyond Reproducibility



Snakemake

Figure: **>370k** downloads since 2015
>1300 citations
>7 citations per week since 2021

The Catalogue

- extremely feature rich

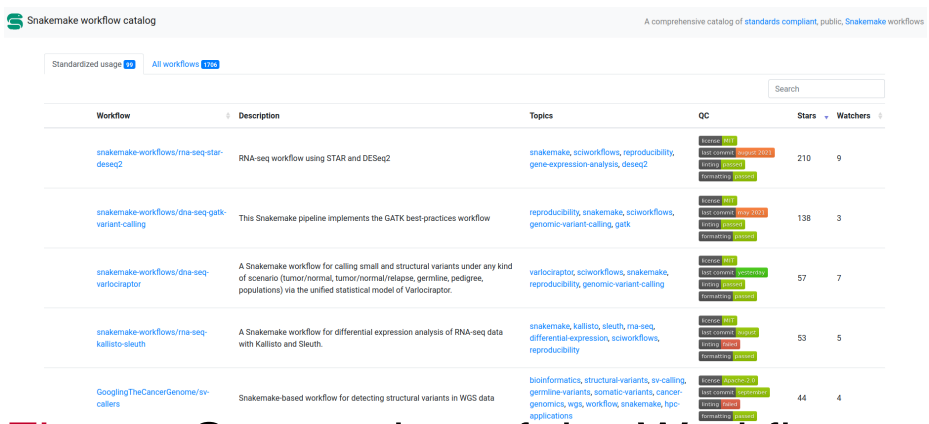


Figure: Screenshot of the Workflow Catalogue

The Catalogue

- extremely feature rich
- over 1800 workflows in its catalogue

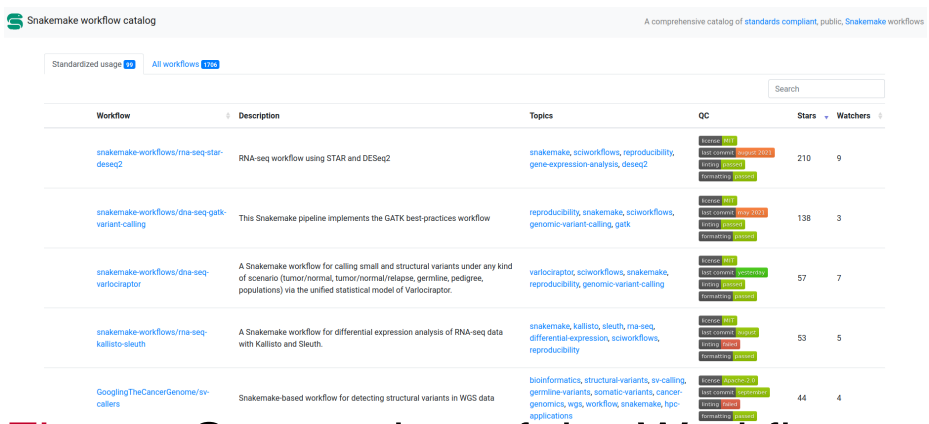


Figure: Screenshot of the Workflow Catalogue

The Catalogue

- extremely feature rich
- over 1800 workflows in its catalogue
- almost a hundred standardized (meaning: will documented and with automatic deployment)

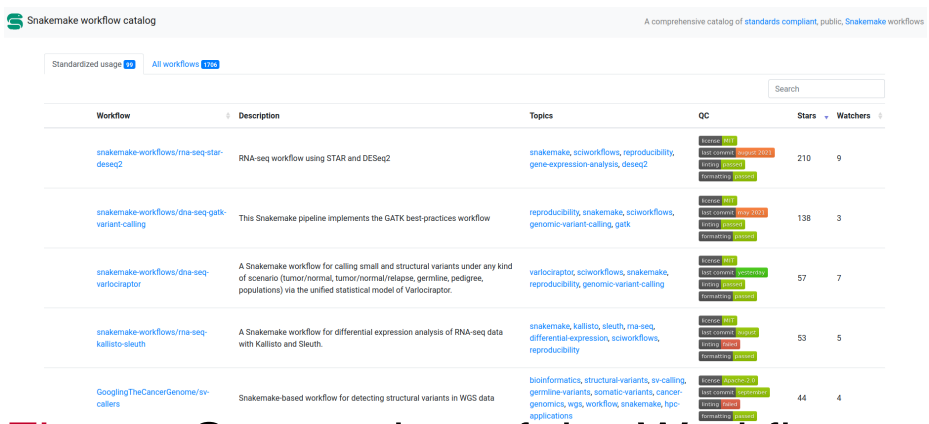


Figure: Screenshot of the Workflow Catalogue

The Catalogue

- extremely feature rich
- over 1800 workflows in its catalogue
- almost a hundred standardized (meaning: will documented and with automatic deployment)
- cluster batch systems are supported (and support for various cloud systems, too)

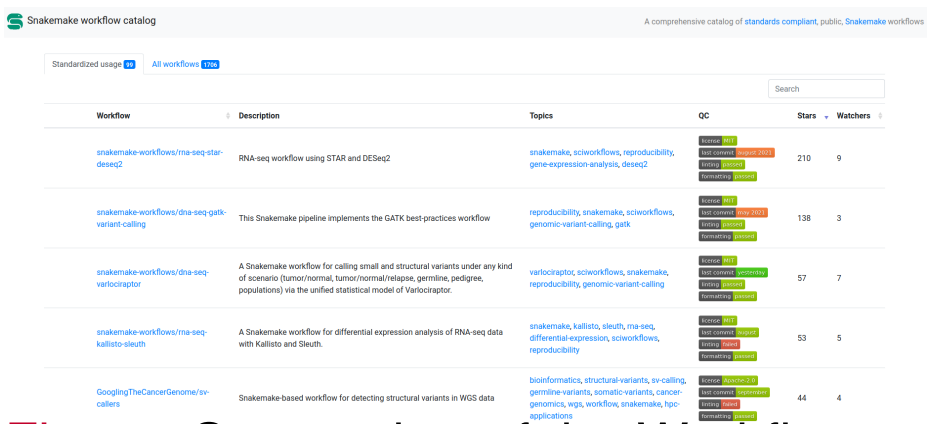


Figure: Screenshot of the Workflow Catalogue

The Catalogue

- extremely feature rich
- over 1800 workflows in its catalogue
- almost a hundred standardized (meaning: will documented and with automatic deployment)
- cluster batch systems are supported (and support for various cloud systems, too)
- there is an option to include Nextflow wrappers, too.

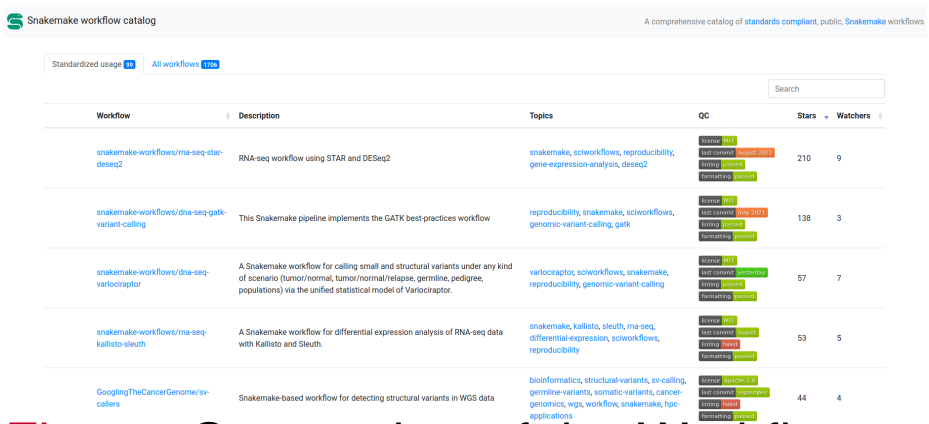


Figure: Screenshot of the Workflow Catalogue

Questions

? The questions you most probably have when starting your Analysis:

- How to start quickly (with the lowest amount of overhead)?
- What are the necessary tools?

? Our question to you:

How do you get this information? And: Is reproducibility and sustainability your concern?

Questions

? The questions you most probably have when starting your Analysis:

- How to start quickly (with the lowest amount of overhead)?
- What are the necessary tools?

? Our question to you:

How do you get this information? And: Is reproducibility and sustainability your concern?

Most frequent Sources

- Your labmate(s)
- The Internet
- Yes, of course ... eventually, when I brag about my paper/thesis.

The Workflow Approach

Workflow Engines answer these questions directly by providing

- entire Workflows can be selected and can be put to action.
- executing routines reliably.

Going HPC

Question

Why would you want to work on a cluster?

Going HPC

Question

Why would you want to work on a cluster?

Answers may include:

- compute power and ressources for big data

Going HPC

Question

Why would you want to work on a cluster?

Answers may include:

- compute power and ressources for big data
- launching scalable (and otherwise portable) workflows with workflow engines

Outline

What is this about?

Questions

- How do I get the software for a particular workflow?
- What is the difference in different build systems and software environments? Why does it matter for me?

Objectives

1. Introducing the "Module" system provided on HPC clusters (briefly).
2. Learning how to install software with "Conda".
3. Knowing how to avoid conflicts between the different software provisioning schemes.

Modules

What is a module?

A module collects all environment variables and settings needed for a particular software package (e. g. path to executable and libraries).

Modules – Command Overview

- List of all available modules

```
$ module avail                    # or 'module av'
```

- Loading a specific module

```
$ module load <modulename> # or 'module add'
```

- Showing all currently loaded modules

```
$ module list
```

- Unloading a specific module

```
$ module unload <modulename>
```

- Unload all active modules

```
$ module purge
```

Modules – looking for specific modules

Looking up modules:

```
$ module spider <search string>
```

Modules – looking for specific modules

Looking up modules:

```
$ module spider <search string>
```

Looking for area specific modules

Try looking for an area specific module, e. g. in “`bwa`”

Modules with easybuild
Or: What is this fuzz at the end of module names?

You will have seen modules like:

```
numlib/FFTW/3.3.10-gompi-2021b
```

Modules with easybuild
Or: What is this fuzz at the end of module names?

You will have seen modules like:

```
numlib/FFTW/3.3.10-gompi-2021b
```

Easybuild Naming Scheme

We are building our modules with easybuild and adopted the following naming scheme for modules:

`<topic>/<name>/<version>-<toolchain>-<toolchain-version>`

Modules with easybuild
Or: What is this fuzz at the end of module names?

You will have seen modules like:

```
numlib/FFTW/3.3.10-gompi-2021b
```

Easybuild Naming Scheme

We are building our modules with easybuild and adopted the following naming scheme for modules:

`<topic>/<name>/<version>-<toolchain>-<toolchain-version>`

Look inside a module to know what will be loaded and set

Do “ module show <module> ”

That's all Folks

Why we will not go in depth now

You can learn more about modules in 101-HPC courses. Later, we will learn how to use `Snakemake` workflows, particularly curated ones, available on the web. We *could* re-write and adapt them for Mogon, it is better to only parameterize them for Mogon and do leave the workflow itself unaltered. This is less cumbersome and as workflow systems, including `Snakemake`, rely on Conda, we will have an in-depth intro to Conda, instead.

Do not mix Conda with Modules

Do not mix Conda with module files - particularly, avoid writing `module load` commands in your `~/.bashrc` file.
Whenever your modules or Conda are using conflicting compilers or environments, you might not be able to execute your software or – *worse* – will result in funny crashes with apparently no reason.

Your Work Environment with Conda

Conda vs. Module Files

Background Module Files

- module files provide an environment per software
- the software usually is compiled on the machine (optimized)
- due to differences in cluster naming schemes and setups portability cannot be granted

Background Conda

- Conda is a machine independent package management systems
- packaged software is provided pre-compiled (NOT optimized)
- Conda allows for grouping software stacks in environments, therefore ensuring portability

Installing Conda

You *could* run

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-
    latest-Linux-x86_64.sh
```

to retrieve (Mini)Conda (a flavour of conda with less overhead).

Hint

URL is from `https://docs.conda.io/en/latest/miniconda.html`. However, we are going to use tweaked scripts provided to you.

Installing Conda

You *could* run

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-
    latest-Linux-x86_64.sh
```

to retrieve (Mini)Conda (a flavour of conda with less overhead).

Hint

URL is from `https://docs.conda.io/en/latest/miniconda.html`. However, we are going to use tweaked scripts provided to you.

Hint

However, instead of downloading, we will work through this together on the slides to come.

We will favour *µ-Mamba* over Conda!

Many know “Conda” as a package manager – and this is correct! But:

Why we recommend using “µ-Mamba”

Conda carries some overhead. Alternative implementations can be faster and require less files. Mamba is a “drop-in” for Conda. This means: *every command is the same*, except we will write `mamba` where usuall `conda` would be.

Why?

Mamba is an implementation of Conda, written in C++. It is able to carry out some tasks in parallel and works considerably faster. In turn, µ-Mamba is a staticically compiled version of Mamba and does not require a “base” environment (we will learn about environments, soon), which means even less overhead.

Installing ~~Conda~~/µ-Mamba

Please run

```
$ "${SHELL}" <(curl -L micro.mamba.pm/install.sh)
```

Installing ~~Conda~~/µ-Mamba - II

You need to confirm (with “Enter”)

- the binary folder.

The tool will tell up about the modification of your `.bashrc` file (which is executed upon *every* login, we will discuss this in a minute).

Installing ~~Conda~~/µ-Mamba - II

You need to confirm (with “Enter”)

- the binary folder.
- the shell you are using (just to be on the save side)

The tool will tell up about the modification of your `.bashrc` file (which is executed upon *every* login, we will discuss this in a minute).

Installing ~~Conda~~/µ-Mamba - II

You need to confirm (with “Enter”)

- the binary folder.
- the shell you are using (just to be on the save side)
- whether conda-forge shall be configured (this is a major resource channel)

The tool will tell up about the modification of your `.bashrc` file (which is executed upon *every* login, we will discuss this in a minute).

Installing ~~Conda~~/µ-Mamba - II

You need to confirm (with “Enter”)

- the binary folder.
- the shell you are using (just to be on the save side)
- whether conda-forge shall be configured (this is a major resource channel)
- the prefix location (where µ-Mamba shall be placed)

The tool will tell up about the modification of your `.bashrc` file (which is executed upon *every* login, we will discuss this in a minute).

Installing ~~Conda~~/µ-Mamba - III

You now have a section like this in your "`~/.bashrc`":

```
# >>> mamba initialize >>>
# !! Contents within this block are managed by
      'mamba init' !!
export MAMBA_EXE='/home/cm/.local/bin/
    micromamba';
export MAMBA_ROOT_PREFIX='/home/cm/micromamba'
    ;
__mamba_setup="$("$MAMBA_EXE" shell hook --
    shell bash --root-prefix "
    $MAMBA_ROOT_PREFIX" 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__mamba_setup"
else
    alias micromamba="$MAMBA_EXE"  # Fallback
        on help from mamba activate
fi
unset __mamba_setup
# <<< mamba initialize <<<
```

Every time you log-in this will be executed. Also, here, "`<prefix>`" denotes *your* prefix.

Installing ~~Conda~~/µ-Mamba - III

You now have a section like this in your "`~/.bashrc`":

```
# >>> mamba initialize >>>
# !! Contents within this block are managed by
      'mamba init' !!
export MAMBA_EXE='/home/cm/.local/bin/
    micromamba';
export MAMBA_ROOT_PREFIX='/home/cm/micromamba'
    ;
__mamba_setup="$("$MAMBA_EXE" shell hook --
    shell bash --root-prefix "
    $MAMBA_ROOT_PREFIX" 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__mamba_setup"
else
    alias micromamba="$MAMBA_EXE"  # Fallback
        on help from mamba activate
fi
unset __mamba_setup
# <<< mamba initialize <<<
```

Every time you log-in this will be executed. Also, here, "`<prefix>`" denotes *your* prefix.

Please edit your "`~/.bashrc`" file and put part in a function, to re-gain manual control:

```
function conda_initialize {
# >>> mamba initialize >>>
# !! Contents within this block are managed by
      'mamba init' !!
export MAMBA_EXE='/home/cm/.local/bin/
    micromamba';
export MAMBA_ROOT_PREFIX='/home/cm/micromamba'
    ;
__mamba_setup="$("$MAMBA_EXE" shell hook --
    shell bash --root-prefix "
    $MAMBA_ROOT_PREFIX" 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__mamba_setup"
else
    alias micromamba="$MAMBA_EXE"  # Fallback
        on help from mamba activate
fi
unset __mamba_setup
# <<< mamba initialize <<<
}
```

Add the highlighted lines! and your login will be faster! Also, this allows to

Why this function in your `.bashrc`?

Documentation

- *every* time you log in, the code in your `.bashrc` will be exectud. Depeding on your conda setup, this can be incredibly slow (another reason to use Mamba or µ-Mamba).

Why this function in your `.bashrc`?

Documentation

- *every* time you log in, the code in your `.bashrc` will be exectud. Depeding on your conda setup, this can be incredibly slow (another reason to use Mamba or µ-Mamba).
- automatic inclusion of Conda/Mamba might cause interference with modules

Why this function in your `.bashrc`?

Documentation

- *every* time you log in, the code in your `.bashrc` will be exectud. Depeding on your conda setup, this can be incredibly slow (another reason to use Mamba or µ-Mamba).
- automatic inclusion of Conda/Mamba might cause interference with modules
- Now, you can run `conda_initialize` in the login shell, jobs scripts, etc. upon demand and deactivate if needed.

Initializing Conda & Mamba

To initialize Conda, simply run

```
$ bash
```

or (better)

```
$ source ~/.bashrc
```

Subsequently, run:

```
$ conda_initialize # if you have this function
```

Reducing Search Overhead - the `.condarc`-File

On many HPC clusters the number of Conda channels (the repositories) is reduced by the whitelisting. Nevertheless, it helps to reduce the search time with a resource file, including a number of definitions, *before* starting:

```
$ cat .condarc
create_default_packages:
  - setuptools # since Python is often needed
# these channels cover most of the required software
channels:
  - conda-forge
  - bioconda
  - defaults
  - r
proxy_servers: # in case a proxy server is present
  http: http://webproxy.zdv.uni-mainz.de:8888
ssl_verify: false
auto_update_conda: false
always_yes: true # avoid confirmation(s)
env_prompt: '($(basename {default_env})) '
```

To obtain the same resource file, run:

```
$ cp ../setup/condarc ~/.condarc
```

More on `.condarc` on the official Conda documentation site

Searching Software with Conda v. Mamba

First you might want to look for software. This is done with

```
$ micromamba search <softwarename>
```

Searching Software with Conda v. Mamba

First you might want to look for software. This is done with

```
$ micromamba search <softwarename>
```

Task

Try this with a software which comes to mind.

Searching Software with Conda v. Mamba

First you might want to look for software. This is done with

```
$ micromamba search <softwarename>
```

Task

Try this with a software which comes to mind.

This will list packages with channel and version information, e. g.

```
$ micromamba search minimap
<snip>
Loading channels: done
# Name                                    Version           Build  Channel
minimap                                   0.2_r124               0  bioconda
minimap                                   0.2_r124       h5bf99c6_4  bioconda
....
```

Installing Software *with* Conda & Mamba - Using Environments

Hint

It is a good habit to have

- *one* environment per workflow
- the environment named as the workflow
- this way, we have a bundle of tools, activate the environment for it
- `snakemake` workflows will install the tools you need for a particular workflow - only *if* these tools are still missing

Note

For now we will not use `snakemake` to install our software. This topic will be covered later.

Conda / Mamba Environments

With Conda/Mamba, we can activate and deactivate environments, which bundle our software. E. g. a software stack per workflow to ensure reproducible runs.

Conda / Mamba Environments

With Conda/Mamba, we can activate and deactivate environments, which bundle our software. E. g. a software stack per workflow to ensure reproducible runs. To generate a new we can run

```
$ micromamba create -n <environment_name>
```

This will create a new environment with a given name in your home directory. Alternatively, i. e. when dealing with file number quotas, you can point the environment to a different filesystem, e. g.:

```
$ micromamba create \
>   -p /path/to/project/fs/<environment_name>
```

Hands On I – Creating your own Environment

Now, create your first environment. We will

- work in the `HOME` directory with
- a *named* environment
- use the downloaded `environment.yaml` file to define the software stack
- pre-compile all Python files with the `--pyc` flag

```
$ micromamba create --pyc -f environment.yaml -n
    snakemake-tutorial
```

Listing Environments

Over time you will have several environments. (So far, there is one.)
To list them all, simply run:

```
$ micromamba env list
```

Creating, Activating, Deactivating Environments

We can create, activate and deactivate environments with

```
$ micromamba create [other args] -n <environment name>
$ micromamba activate <environment name>
$ micromamba deactivate
```

Task

Activate the environment you just created!

Interlude – Managing your Prompt

Question

Did you notice the last line in your `.condarc` file? What does it do?

Remember:

```
$ cat .condarc
....
env_prompt: '($(basename {default_env})) '
```

Interlude – Managing your Prompt

? Question

Did you notice the last line in your `.condarc` file? What does it do?

Remember:

```
$ cat .condarc
....
env_prompt: '($(basename {default_env})) '
```

It causes an unnamed environment prompt to shrink, e. g.:

```
(my_env) user@host:directory $
# instead of
(/some/long/path/my_env) user@host:directory $
```

Outline

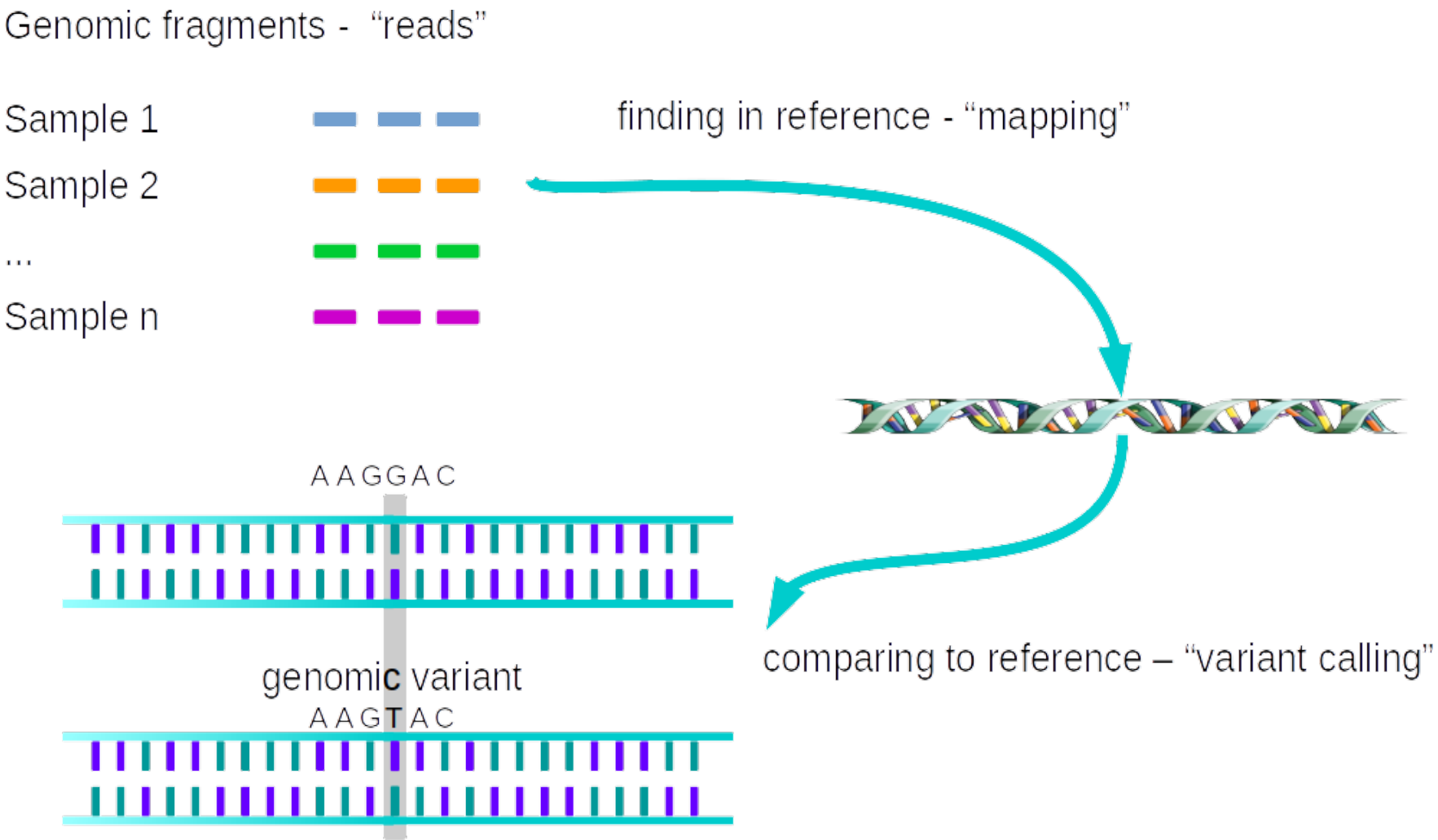
What is this about?

Questions

- How can workflow development be started?
- What is a good directory layout?
- How does a workflow description look like?

Objectives

1. Introduce you to conceptualization
2. First demonstration of directory layouts
3. Introducing a first (very basic) workflow


Genomic fragments - "reads"
Sample 1
Sample 2
...
Sample n
finding in reference - "mapping"
AAGGAC
genomic variant
AAGTAC
comparing to reference – "variant calling"

Getting the Tutorial Data

This is Subject to Change!

This tutorial carries a bug in `bioconda`: A certain piece of software is not compatible with the current version of Python. Hence, we run a script (we will look into it), which downloads our data and downgrades Python, such that everything will be working.
Of course, this might all not be necessary in a future course.

Getting the Tutorial Data

This is Subject to Change!

This tutorial carries a bug in `bioconda`: A certain piece of software is not compatible with the current version of Python. Hence, we run a script (we will look into it), which downloads our data and downgrades Python, such that everything will be working.
Of course, this might all not be necessary in a future course.

For the tutorial we run:

```
$ bash setup/02_tutorial_setup.sh
```

The Setup Script

```
# 0. we are going to use ONE environment with the same
#    name as the workflow we are going to create
NAME="snakemake-tutorial"

# 1. On many HPC systems we are going to install via a proxy.
#    This might ignore TLS, hence we turn off all warnings.
#    Note: TLS might still be used, but not ALL proxies use it.
export PYTHONWARNINGS="ignore:Unverified HTTPS request"

# 1. create the environment for the tutorial
mamba create --name $NAME

# 2. activate the environment
mamba activate $NAME

# TODO: This step is necessary because of the outdated
#       setup, where Python >= 3.10 does not work
#       with pysam
mamba install Python=3.9
```

Here we create an environment, activate it and install an older version of Python. Questions to the code?

Getting the example Data

Again we are using a script to download and extract our example data. Please run

```
$ bash setup/03_get_data.sh
```

Subsequently change into the directory `snakemake-tutorial` which has been created.

Getting the example Data

Again we are using a script to download and extract our example data. Please run

```
$ bash setup/03_get_data.sh
```

Subsequently change into the directory `snakemake-tutorial` which has been created.

Question

Which files do you find?

The Collector Script

```
#!/bin/bash

# 1. we are going to work in a dedicated directory:
mkdir snakemake-tutorial && cd snakemake-tutorial

# 2. download the sample data
echo "downloading sample data"
curl -L https://api.github.com/repos/snakemake/snakemake-tutorial-data/tarball -o snakemake-tutor

# 3. unpack the sample data
echo "unpacking sample data"
tar --wildcards -xf snakemake-tutorial-data.tar.gz --strip 1 "*/data" "*/environment.yaml"

# 4. this is necessary, because we need a simple list of packages
#    after we downgraded the python version in the step afore.
cat environment.yaml | tail -n+5 | cut -c5- > environment2.txt

# 5. install all necessary software
mamba install --file environment2.txt
```

Two Notes

1. The tinkering in step 4 is so weird, due to the reasons explained.
2. You can install software from a) a simple text file, b) during creation of the environment and c) let `snakemake` do it for you (introduced later).

Installing Software *with* Conda - Using Environments II

Using our new environment we can activate it:

```
$ mamba activate snakemake-tutorial
```

and install "`snakemake`" by running

```
$ mamba install snakemake
```

Now, we are all set.

Hint

Just that you know: "`snakemake`" is able to install the requested software for you!

We are almost there . . .

In order to start working, we need an editor.

- load your editor `geany` or `gedit` *before* attempting to load modules in the same shell. Start it like

```
$ geany &
```

We are almost there . . .

In order to start working, we need an editor.

- load your editor `geany` or `gedit` *before* attempting to load modules in the same shell. Start it like

```
$ geany &
```

- Best use a terminal multiplexer **or** multiple logins (otherwise you have to develop and execute in one shell).

Outline

What is this about?

Questions

How do I write a simple workflow?

Objectives

1. Understand the components of a Snakefile: rules, inputs, outputs, and actions.
2. Write a simple Snakefile.
3. Run Snakemake from the shell.

Before we begin . . .

Working with closure Files

To ease the excercises and save typing time, all excercises are supplied as cloze texts.
`Snakemake` relies on a file called `Snakefile` to be present. You can either rename your cloze texts like

```
$ cp <number>_Snakefile Snakefile
```

or specify the workflow file on the commandline with an additional flag:

```
$ snakemake <other arguments> \
> --snakefile <number>_Snakefile
```

Also note: `\` may denote a linebreak in `Bash` and `>` its continuation. Omit these and you have a one-liner. It merely serves to fit text on screen, here.

Layout of a Workflow Development Directory

The idea is to have a neat overview:

```
workflow folder
├── scripts
│   ├── some
│   │   └── scriptfile.py
│   ├── some
│   │   └── scriptfile.sh
│   └── some
│       └── scriptfile.R
├── Snakefile
└── and more ...
```

Hint

Our long term goal:
Have a orderly overview and seperation of data and code. We will start with one `Snakefile`.

Our first `Snakefile`!

Our first “ `rule` ”; we want to map reads onto a reference genome.

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

Documentation

This code is build file, which for `Snakemake` is called a `Snakefile` – a file executed by `Snakemake`.

Our first `Snakefile`!

We are going to “map” our reads onto the genome.

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

For this we are using `bwa` , specifically `bwa mem` . We call our rule accordingly `bwa_map` .

Our first `Snakefile`!

`input`, `output` and `shell` are called “directives”:

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

The `input` and `output` directives are followed by lists of files that are expected to be used or created by the rule. In the simplest case, these are just explicit Python strings.

Our first `Snakefile`!

The `shell` may be one line or contain line breaks:

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

Python will concatenate those Strings! Be sure to include spaces to make up a valid command in Bash.

Our first `Snakefile`!

Be sure to have copied everything to your `Snakefile` and save it.

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

You will find the working content in `01_Snakefile`, too.

`Snakefile`s are Python Files

Some Background

1. Like Python, you can use either tabs or spaces for indentation — don’t mix! Consensus is to only use spaces.
2. Together, the target, dependencies, and actions form a rule. A rule is a recipe for how to make things.

Testing our `Snakefile`

When a workflow is executed, `Snakemake` tries to generate given target files. Target files can be specified via the command line. By executing

```
$ snakemake -np mapped_reads/A.bam
```

in the working directory containing the `Snakefile`, we tell `Snakemake` to generate the target file `mapped_reads/A.bam`.
We are using

- `–n/––dry–run` to show the *planned* execution and

Testing our `Snakefile`

When a workflow is executed, `Snakemake` tries to generate given target files. Target files can be specified via the command line. By executing

```
$ snakemake -np mapped_reads/A.bam
```

in the working directory containing the `Snakefile`, we tell `Snakemake` to generate the target file `mapped_reads/A.bam`.
We are using

- `-n/--dry-run` to show the *planned* execution and
- `-p` to print the intended shell command.

Finally - Running `Snakemake`!

Now, we can run `Snakemake`:

```
$ snakemake --cores=1 mapped_reads/A.bam
```

Note:

The `--cores=1` is necessary, because we are executing “locally” and `Snakemake` would like to know how much of the resources we may use (you can try without, though, to see what happens).

You should see the expected output (`ls`) and lines which reads:

```
1 of 1 steps (100%) done
Complete log: .snakemake/log/2022-09-17T145657.968906.
    snakemake.log
```

Re-Running Workflows

Try to run `Snakemake` again:

```
$ snakemake --cores=1 mapped_reads/A.bam
```

Re-Running Workflows

Try to run `Snakemake` again:

```
$ snakemake --cores=1 mapped_reads/A.bam
```

Oops:

```
Nothing to be done (all requested files are present and up to
    date).
```

Re-Running Workflows

Try to run `Snakemake` again:

```
$ snakemake --cores=1 mapped_reads/A.bam
```

Oops:

```
Nothing to be done (all requested files are present and up to
    date).
```

Now, do:

```
$ touch mapped_reads/A.bam
```

And run `Snakemake` once more.

Question

What happens? Why?

When do Rules get executed? - The Solution

When it is asked to build a target, `Snakemake` checks the “last modification time” of both the *target* and its *dependencies*. If either

- any dependency has been updated
- or a job failed to produce a target (completely)

upon re-run `Snakemake` will only rebuild the files that, either directly or indirectly, depend on the file that changed. This is called an *incremental build*.

When do Rules get executed? - The Solution

When it is asked to build a target, `Snakemake` checks the “last modification time” of both the *target* and its *dependencies*. If either

- any dependency has been updated
- or a job failed to produce a target (completely)

upon re-run `Snakemake` will only rebuild the files that, either directly or indirectly, depend on the file that changed. This is called an *incremental build*.

Documentation

By explicitly recording the inputs to and outputs from steps in our analysis and the dependencies between files, Snakefiles act as a type of documentation, reducing the number of things we have to remember.

Too much Typing!1!11!

A closer look on our first rule reveals:

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem data/genome.fa"
        " data/samples/A.fastq"
        " | samtools view -Sb - >"
        " mapped_reads/A.bam"
```

Way too much redundancy!

Introducing Wildcards

`input` and `output` can be referenced in our file, with an expression called “wildcard”:

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem {input}"
        " | samtools view -Sb - >"
        " {output}"
```

Since the rule has multiple input files, `Snakemake` will concatenate them, separated by a whitespace. In other words, `Snakemake` will replace `{input}` with `data/genome.fa data/samples/A.fastq` before executing the command.
A working example can be found in `02_Snakefile`.

Further Workflow Decoration

`Snakemake` allows generalizing rules by using named wildcards. Simply replace the `A` in the second input file and in the output file with the wildcard `{sample}`, to yield:

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"
```

When `Snakemake` determines that this rule can be applied to generate a target file by replacing the wildcard `{sample}` in the output file with an appropriate value, it will propagate that value to all occurrences of `{sample}` in the input files and thereby determine the necessary input for the resulting job.

One Caveat!

Multiple Wildcards

You can have multiple wildcards in your file paths, however, to avoid conflicts with other jobs of the same rule, *all output files* of a rule *have to contain exactly the same wildcards*.

Execution

When executing

```
$ snakemake -np mapped_reads/B.bam
```

`Snakemake` will determine that the rule `bwa_map` can be applied to generate the target file by replacing the wildcard `{sample}` with the value `B`.
To analyse samples `A` and `B`, we can specify two targets

```
$ snakemake -np mapped_reads/A.bam mapped_reads/B.bam
```

or use Bash’s brace expansion

```
$ snakemake -np mapped_reads/{A,B}.bam
```

Sorting Alignments

For later steps, we need the read alignments in the BAM files to be sorted. This can be achieved with the `samtools` `sort` command. We add the following rule beneath the `bwa_map` rule:

```
rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"
```

Task

Please refer to your template `03_Snakefile`.

Using Wildcards

The rule will take input file from the `mapped_reads` directory and store a sorted version in the `sorted_reads` directory.

Using Wildcards

The rule will take input file from the `mapped_reads` directory and store a sorted version in the `sorted_reads` directory.

Documentation

`Snakemake` will automatically create missing directories!

You noticed `–T sorted_reads/{wildcards.sample}` ?

Using Wildcards

The rule will take input file from the `mapped_reads` directory and store a sorted version in the `sorted_reads` directory.

Documentation

`Snakemake` will automatically create missing directories!

You noticed `-T sorted_reads/{wildcards.sample}`? For sorting, `samtools` requires a prefix specified with the flag `-T`. Here, we need the value of the wildcard `sample`. `Snakemake` allows to access wildcards in the shell command via the wildcards object that has an attribute with the value for each wildcard.

Execution

When you run

```
$ snakemake -np sorted_reads/B.bam
```

you will notice that `Snakemake` wants to run the first rule `bwa_map` and then the rule `samtools_sort` to create the desired target file.

Documentation

Dependencies are resolved automatically by matching file names.

Indexing Read Alignments

We need to use samtools again to index the sorted read alignments so that we can quickly access reads by the genomic location they were mapped to. This can be done with the following rule:

```
rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"
```

Note

You will find this code in the next cloze text. It is not worth proceeding without the next step.

Calling Genomic Variants - Background I

The next step in our workflow will aggregate the mapped reads from all samples and jointly call genomic variants on them. We need two tools: samtools and bcftools.

Calling Genomic Variants - Background I

The next step in our workflow will aggregate the mapped reads from all samples and jointly call genomic variants on them. We need two tools: samtools and bcftools.
`Snakemake` provides a helper function for collecting input files that helps us to describe the aggregation in this step. With

```
expand("sorted_reads/{sample}.bam", sample=SAMPLES)
```

we obtain a list where the given pattern `sorted_reads/{sample}.bam` was formatted with the values in a given list of samples `SAMPLES`, i. e.

```
["sorted_reads/A.bam", "sorted_reads/B.bam"]
```

Calling Genomic Variants - Background II

This is particularly useful when dealing with *multiple* wildcards, e. g.:

```
expand("sorted_reads/{sample}.{replicate}.bam",
       sample=SAMPLES, replicate=[0, 1])
```

With all elements of `SAMPLES` and the list `[0, 1]`, we get:

```
["sorted_reads/A.0.bam", "sorted_reads/A.1.bam",
 "sorted_reads/B.0.bam", "sorted_reads/B.1.bam"]
```

Calling Genomic Variants

Later, we will learn how to provide input more sophistically . . .
For now, we will define a list on top of our `Snakefile`:

```
SAMPLES = ["A", "B"]
```

Now, we can add the following rule to our `Snakefile`:

```
rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam", sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai", sample=SAMPLE
    output:
        "calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | "
        "bcftools call -mv - > {output}"
```

This is part of `05_Snakefile`.

Outline

What is this about?

Questions

- How can I automatically manage dependencies and outputs?
- How can I use Python code to add features to my pipeline?

Objectives

1. Use variables, functions, and imports in a Snakefile.
2. Learn to use the `run` action to execute Python code as an action.

Why Python in `Snakefile`?

Sometimes we do *not* want to run 3rd party code, but run the occasional script for data manipulation or plotting or . . .

Why Python in `Snakefile`?

Sometimes we do *not* want to run 3rd party code, but run the occasional script for data manipulation or plotting or . . .

Documentation

`Snakefile`s are Python code (albeit special Python code)!)

Using functions in Snakefiles

We already met the `expand()` -function. Let's revisit it in detail!

Task

Start Python and follow along!

```
from snakemake.io import expand, glob_wildcards
```

`expand()` is used frequently, to expand a Snakemake wildcard(s) into a set of filenames:

```
>>> expand('folder/{wildcard1}_{wildcard2}.txt',
...        wildcard1=['a', 'b', 'c'],
...        wildcard2=[1, 2, 3])
```

Using functions in `Snakefile`s II

Question

What is the result?

Using functions in `Snakefile`s II

Question

What is the result?

Answer:

Every permutation of `wildcard1` and `wildcard2`!

Using functions in `Snakefile`s II

Question

What is the result?

Answer:

Every permutation of `wildcard1` and `wildcard2`!

Let us try `glob_wildcards()`, now:

```
glob_wildcards('data/samples/{replicas}.fastq').replicas
```

Using functions in `Snakefile`s II

Question

What is the result?

Answer:

Every permutation of `wildcard1` and `wildcard2`!

Let us try `glob_wildcards()`, now:

```
glob_wildcards('data/samples/{replicas}.fastq').replicas
```

Wow! So, easy!

Question

What happend? What happens if you leave `.replicas` away?

`expand()` vs. `glob_wildcards()`

The Difference

- `expand()` is usefull to yield all permutations of wildcards
- `expand()` is oblivious to the underlying file system
- `glob_wildcards` infers files from wildcards
- `glob_wildcards` must be used with care - users tend to name files arbitrarily

Interlude – Meeting the `run` Action

It is possible to run Python snippets in `Snakemake`. For this, we have to replace the `shell:` action by a `run:` action:

Task

Imaging this code in a `Snakefile`:

```
# at the top of the file
import glob

# add this wherever
rule print_sample_names:
    run:
        print('These are all the samples names:')
        for sample in glob.glob('data/samples/*.fastq'):
            print(sample)
```

Interlude – Meeting the `run` Action

It is possible to run Python snippets in `Snakemake`. For this, we have to replace the `shell:` action by a `run:` action:

Task

Imaging this code in a `Snakefile`:

```
# at the top of the file
import glob

# add this wherever
rule print_sample_names:
    run:
        print('These are all the samples names:')
        for sample in glob.glob('data/samples/*.fastq'):
            print(sample)
```

The idea:

This allows for Python snippets in rules without saving Python files, e.g. for short data mangling or plotting.

Snakemake Specials for Python

Remember this part:

```
shell: 'some_command -i {input} -o {output}'
```

Question

Isn't this rather tedious, considering that Snakemake is written in Python? Could we forward rule specific information to our scripts?

Snakemake's features for external Scripts

In addition to the `shell` and `run` directives, Snakemake offers a `script` directive, too:

```
rule NAME:
    input:
        "path/to/inputfile",
    output:
        "path/to/outputfile",
    script:
        "scripts/script.py"
```

This lets you use (in `script.py`):

```
def do_something(data_path, out_path):
    # python code

do_something(snakemake.input[0], snakemake.output[0])
```

Interlude – The same feature is offered for R

```
do_something <- function(data_path, output_path) {
    # R code
}

do_something(snakemake@input[[1]], snakemake@output[[1]])
```

Interlude – Debugging external Scripts

Debugging Python Scripts

To debug Python scripts, you can invoke `Snakemake` with

```
$ snakemake --debug
```

As always in Python, you may use `print()` and `logging()` functions which may write to a log file during the run to check whether the script behaves as expected.

Interlude – Debugging external Scripts

Debugging Python Scripts

To debug Python scripts, you can invoke `Snakemake` with

```
$ snakemake --debug
```

As always in Python, you may use `print()` and `logging()` functions which may write to a log file during the run to check whether the script behaves as expected.

Debugging R Scripts

To debug R scripts, you can save the workspace with `save.image()`, and invoke R after `Snakemake` has terminated.

Preparing the next exercise:

We want to plot our Results!

Add the following rule to your `Snakefile` we could plot our variant statistics:

```
rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

It uses the `script` directive and expects a directory `scripts` relative to our `Snakefile`.

We want to plot our Results!

Add the following rule to your `Snakefile` we could plot our variant statistics:

```
rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

It uses the `script` directive and expects a directory `scripts` relative to our `Snakefile`.

Task

Create the missing directory.

Hands On II – Adding the Plotting Script

Task

Open an editor and follow along.

First, we start with importing a plotting library; save the script under `scripts/plot-quals.py`:

```
import matplotlib
```

Hands On III – Adding the Plotting Script

Task

Open an editor and follow along.

We enforce writing to suppress the interactive mode:

```
import matplotlib
matplotlib.use("Agg")
```

Hands On IV – Adding the Plotting Script

Task

Open an editor and follow along.

We want an alternative interface:

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
```

Hands On V – Adding the Plotting Script

Task

Open an editor and follow along.

...and need to import a function to read our variant file:

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile
```

Hands On VI – Adding the Plotting Script

Task

Open an editor and follow along.

Let's start a list comprehension ...

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [
```

Hands On VII – Adding the Plotting Script

Task

Open an editor and follow along.

. . . to construct a list of quality scores from our previous outputs:

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [record.qual for record
         in VariantFile(snakemake.input[0])]
```

Hands On VIII – Adding the Plotting Script

Task

Open an editor and follow along.

Finally, plot a histogram ...

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [record.qual for record
         in VariantFile(snakemake.input[0])]

plt.hist(quals)
```

Hands On IX – Adding the Plotting Script

Task

Open an editor and follow along.

...and save the figure.

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [record.qual for record
         in VariantFile(snakemake.input[0])]

plt.hist(quals)
plt.savefig(snakemake.output[0])
```

Hands On X – Adding the Plotting Script

Task

Open an editor and follow along.

This should be our final script:

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [record.qual for record
         in VariantFile(snakemake.input[0])]

plt.hist(quals)
plt.savefig(snakemake.output[0])
```

Outline

What is this about?

Questions

So far our workflow will not run properly. Why? What do we need to change?

Objectives

1. Understanding the target idea.
2. Running simple workflows.

Why Target Rules

So far, we always executed the workflow by specifying a target file at the command line. How cumbersome!

Documentation

We remember: `Snakemake` will automatically determine for a given rule, which expected outcomes are missing and execute all necessary rules, accordingly.

Why Target Rules

So far, we always executed the workflow by specifying a target file at the command line. How cumbersome!

Documentation

We remember: `Snakemake` will automatically determine for a given rule, which expected outcomes are missing and execute all necessary rules, accordingly.
The "trick" is that a workflow can have a "target" rule, which specifies the *final* output(s) of a workflow. Any invokation of `Snakemake` will then execute *all* rules of a workflow.

Best Practice

Documentation

If no target is given at the command line, `Snakemake` will define the first rule of the `Snakefile` as the target.

Conventially, this rule is named `all`. This means that we add a rule at the top of our workflow:

Question

Which is our target file?

```
rule all:
    input:
```

Best Practice

Documentation

> If no target is given at the command line, `Snakemake` will define the first rule of the `Snakefile` as the target.

Conventially, this rule is named `all`. This means that we add a rule at the top of our workflow:
Our target rule is:

```
rule all:
    input:
        "plots/quals.svg"
```

Our “final” Workflow

Refer to `06_Snakefile` as a reference.

```
SAMPLES = ["A", "B"]

rule all:
    input:
        "plots/quals.svg"


rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb"
        " - > {output}"

rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort"
        " -T sorted_reads/{wildcards.sample}"
        " -O bam {input} > {output}"
```

```
rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam",
            sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai
            sample=SAMPLES)
    output:
        "calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa}"
        " {input.bam} | "
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Hands On XI – Executing this Workflow

Does our Workflow contain errors? We run a debug trial:

```
$ snakemake --debug
```

Hands On XII – Executing this Workflow

Does our Workflow contain errors? We run a debug trial:

```
$ snakemake --debug
```

Some targets are already present, we want the entire workflow again:

```
$ snakemake -j4 --forcerun
```

Question

What do you observe? Why `-j4` ?

Hands On XIII – Visualising the Output

On MOGON the linux distro is CentOS (or now AlmaLinux), which provides the `display` -program to display simple images. We shall invoke:

```
$ display plots/quals.svg
```

The figure has no axis-labels.

Question

What does the figure display?

Outline

What is this about?

Questions

- How do we plot our workflows for a publication?

Objectives

1. Learn to plot workflows using `Snakemake` and `GraphViz`.

Hands On XIV – Plotting the Workflow

`Snakemake` has a build-in plotting feature. Run

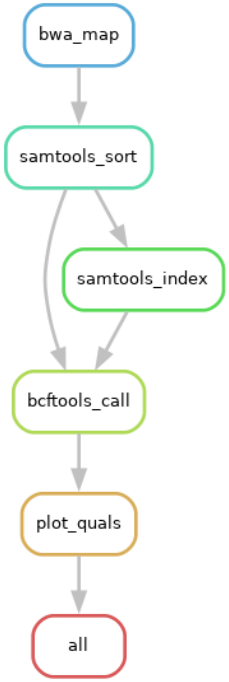
```
$ snakemake --rulegraph | dot -Tpng > <your_workflow.png>
```

to plot your workflow graph. And

```
$ display <your_workflow.png>
```

to display the workflow.

Your DAG:

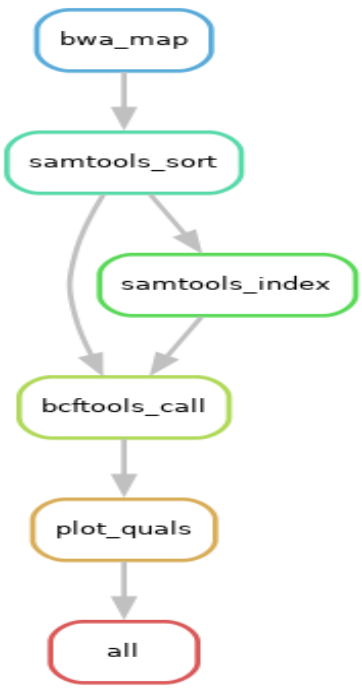

bwa_map
samtools_sort
samtools_index
bcftools_call
plot_quals
all

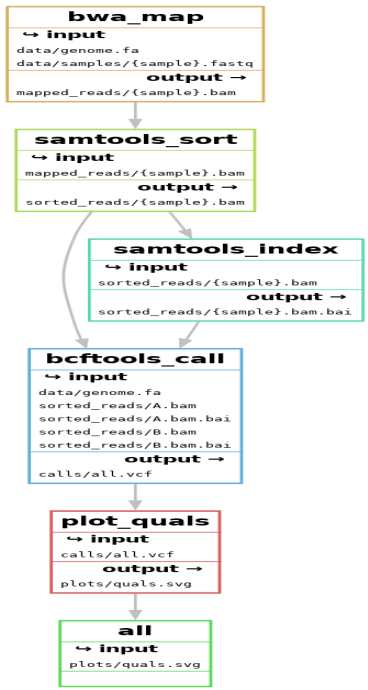
Hands On XV – Produce different Plots

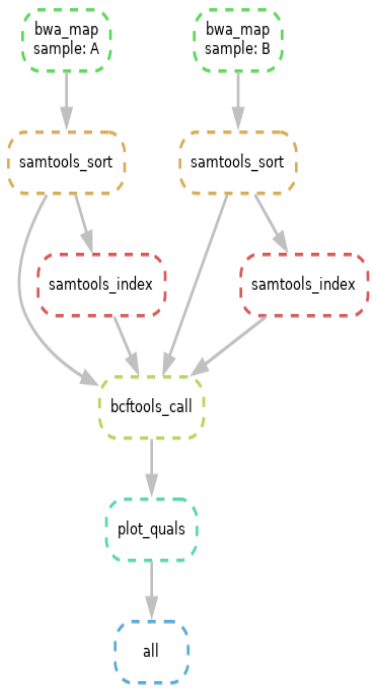
What do you observe, if you try:

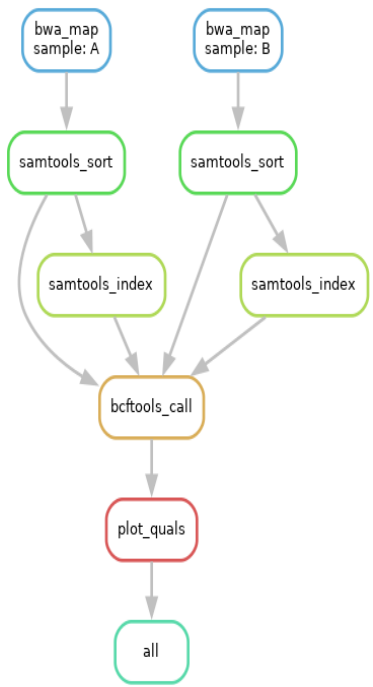
```
$ snakemake --rulegraph | dot -Tpng > <your_rulegraph.png>
$ # or
$ snakemake --filegraph | dot -Tpng > <your_filegraph.png>
$ # or
$ snakemake --dag | dot -Tpng > <your_dag.png>
$ # or --dag again after
$ snakemake --delete-all-output -c1
```

Plot Comparison









(a) `--rulegraph` (b) `--filegraph` (c) `--dag`, with rules done (d) `--dag`, with rules still to do

Outline

What is this about?

Question

- How does ordinary job submission work on a cluster?
- How does it work using Snakemake? (Which parameterization is necessary?)

Objectives

1. Get a feeling for the SLURM batch system.

What is a scheduler?

On HPC systems you do not *just start*, you need a *"scheduler"*. So, what's that?
A scheduler (or "batch system") on a HPC system should...

- provide an interface to help defining workflows and/or job dependencies
- enable automatic submission of executions
- provide interfaces to monitor the executions
- prioritise the execution order of unrelated jobs

Since late spring 2017 we are using SLURM .

Promises, promises and even more promises

How does a scheduler work?

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.
- how many CPUs you will need (and in which combination).

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.
- how many CPUs you will need (and in which combination).
- whether you need something special (e.g. a GPU).

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.
- how many CPUs you will need (and in which combination).
- whether you need something special (e.g. a GPU).

The scheduler will act:

- It will queue up your job (and decide when it will start relative to others).

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.
- how many CPUs you will need (and in which combination).
- whether you need something special (e.g. a GPU).

The scheduler will act:

- It will queue up your job (and decide when it will start relative to others).
- It will decide where your job will run physically (which hosts).

Promises, promises and even more promises

How does a scheduler work?

You tell it...

- how much memory (RAM, scratch space) your job will need.
- how much time you will spend on it.
- how many CPUs you will need (and in which combination).
- whether you need something special (e.g. a GPU).

The scheduler will act:

- It will queue up your job (and decide when it will start relative to others).
- It will decide where your job will run physically (which hosts).
- Eventually it will start your job (if everything was correct).

Hands On XVI – Your first job script

Hint

From now on, we will be scripting examples (cloze-based). For this you will need an editor. If you do not know any other editor, use `gedit`:

```
$ # cd into appropriate directory
$ # Start gedit with the command
$ gedit &
```

Hint

The `&` will put the editor into the background.

Hands On XVI – Your first job script

```
#!/bin/bash

#SBATCH -A m2_jgu-ngstraining
#SBATCH -p smp

srun echo "Hello World from job $SLURM_JOB_ID on node $(hostname)"
```

Task

Save the script as `hello_world.sh` and submit it with the following statement:

```
$ sbatch hello_world.sh
```

Hands On XVI – Your first job script

```
#!/bin/bash

#SBATCH -A m2_jgu-ngstraining
#SBATCH -p smp

srun echo "Hello World from job $SLURM_JOB_ID on node $(hostname)"
```

Important Items and Aspects

- Interpreter directive
- Account necessary
- Reservation only during course
- Job step with `srun`
- Question: Where is the output?

End of HPC Intro

We could co on with *many* details with regards to the scheduler, the file system, etc..

HPC Courses

The HPC teams offers courses to:

- HPC Intro
- Bash Intro
- Research Data Management
- lots of more (hopefully)

End of HPC Intro

We could co on with *many* details with regards to the scheduler, the file system, etc..

HPC Courses

The HPC teams offers courses to:

- HPC Intro
- Bash Intro
- Research Data Management
- lots of more (hopefully)

What's next

We are going to parameterize our workflow**s** for clusters and for our applications in `Snakemake`!

Outline

What is this about?

Questions

- How can a workflow be made really usable?
- How can I avoid changing the workflow source for every new input?

Objectives

1. Learn how to make use of configuration files.
2. Learn how to add parameters to your workflow.

Documentation

`Snakemake` has an extensive command line interface (CLI). *Everything* can be configured on the command line. In addition (almost) everything can be specified in a configuration file.

Documentation

`Snakemake` has an extensive command line interface (CLI). *Everything* can be configured on the command line. In addition (almost) everything can be specified in a configuration file.

Which parameter goes where? Some rules of thumb:

The CLI:

- frequently changing parameters
- short parameters
- default parameters

The Config File:

- non-volantile parameters specific to your analysis (those which merit mentioning in paper should always go into a file)
- long parameters
- otherwise workflow specific parameters

The `Snakemake resources` Section

`Snakemake` rules provide an additional `resource` section:

```
rule <name>:
    ...
    resources:
        partition='parallel',
        mem_mb=1800,
        cpus_per_task=4
```

Hint

Note the ,!

The `Snakemake resources` Section

`Snakemake` rules provide an additional `resource` section:

```
rule <name>:
    ...
    resources:
        partition='parallel',
        mem_mb=1800,
        cpus_per_task=4
```

Hint

Note the ,!

Documentation

You *may* define *any* resource keyword within any rule.

The `Snakemake resources` Section - its Downside

Every Resource Spec needs a Change per Rule???

You might have noticed, this specification per rule is most untidy. `Snakemake`'s design principle is: ship workflows which run *everywhere* & *every time*.

The `Snakemake resources` Section - its Downside

Every Resource Spec needs a Change per Rule???

You might have noticed, this specification per rule is most untidy.
`Snakemake`'s design principle is: ship workflows which run *everywhere* & *every time*.
Relax: Every bit we can specify:

- in a `Snakefile`,
- on the command line,
- and re-usable in configuration files!

The Configuration File

If observed closely you have seen this file:

```
$ tree

  /
  ├── config
  │   └── samples.yaml
  └── other stuff
```

The Configuration File

If observed closely you have seen this file:

```
$ tree
.
/
├── config
│   └── samples.yaml
└── other stuff
```

Documentation

You can store a yaml file with *your* workflow configuration – which may be combined with the desinger's configuration.

The Configuration File

If observed closely you have seen this file:

```
$ tree
.
/
├── config
│   └── samples.yaml
└── other stuff
```

Documentation

You can store a yaml file with *your* workflow configuration – which may be combined with the desinger's configuration.

Warning

It is better to specify fully qualified paths to your data! The tilde is only there to make a point!

Using the Configuration File

To point to the configuration file, you can add a flag, e. g.:

```
$ snakemake ... --configfile ./config/config.yaml
```

Or, alternatively in your `Snakefile`:

```
configfile: "config.yaml"
```

Question

How do pick up resource parameters in out `Snakefile`?!

Working with Configuration Data

Hint

Remember: `Snakefile`s are Python files.

Working with Configuration Data

Hint

Remember: `Snakefile`s are Python files.

Given a file `config.yaml` with contents:

```
samples:
    A: data/samples/A.fastq
    B: data/samples/B.fastq
```

we can read our samples within our workflow using

```
configfile: "config.yaml"

rule bcftools_call:
    input:
        bam=expand("sorted_reads/{sample}.bam",
            sample=config["samples"]),
        ...
```

A Comparison

```
rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam",
                   sample=config["samples"]),
        bai=expand("sorted_reads/{sample}.bam.bai",
                   sample=config["samples"])
```

To

```
rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam",
                    sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai",
                   sample=SAMPLES)
```

? Which are the benefits?

Input Functions

Since we have stored the path to the FASTQ files in the config file, we can also generalize the rule `bwa_map` to use these paths. This case is different to the rule `bcftools_call` we modified above. To understand this, it is important to know that `Snakemake` workflows are executed in three phases.

- In the *initialization phase*, the files defining the workflow are parsed and all rules are instantiated.

Input Functions

Since we have stored the path to the FASTQ files in the config file, we can also generalize the rule `bwa_map` to use these paths. This case is different to the rule `bcftools_call` we modified above. To understand this, it is important to know that `Snakemake` workflows are executed in three phases.

- In the *initialization phase*, the files defining the workflow are parsed and all rules are instantiated.
- In the *DAG phase*, the directed acyclic dependency graph of all jobs is built by filling wildcards and matching input files to output files.

Input Functions

Since we have stored the path to the FASTQ files in the config file, we can also generalize the rule `bwa_map` to use these paths. This case is different to the rule `bcftools_call` we modified above. To understand this, it is important to know that `Snakemake` workflows are executed in three phases.

- In the *initialization phase*, the files defining the workflow are parsed and all rules are instantiated.
- In the *DAG phase*, the directed acyclic dependency graph of all jobs is built by filling wildcards and matching input files to output files.
- In the *scheduling phase*, the DAG of jobs is executed, with jobs started according to the available resources.

Input Functions - II

Why we cannot do the same all over the Workflow

We cannot determine the FASTQ paths for rule `bwa_map` from the config file in this phase, *because we don't even know which jobs will be generated from that rule*. Instead, we need to defer the determination of input files to the DAG phase. This can be achieved by specifying an input function instead of a string as inside of the input directive.

```
def get_bwa_map_input_fastqs(wildcards):
    return config["samples"][wildcards.sample]

rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
```

Hands On XVII – Adding a new Sample

In the `data/samples` folder, there is an additional sample `C.fastq`. Add that sample to the config file and see how `Snakemake` wants to recompute the part of the workflow belonging to the new sample, when invoking with

```
$ snakemake -n --forcerun bcftools_call
```

Copy the file XXX as your `Snakefile`.

Rule Parameters

Warning

Frequently, party programs cannot be used with their defaults, additional parameters are required.

In our workflow, it is reasonable to annotate aligned reads with so-called read groups, that contain metadata like the sample name. To do so, we add to our `bwa_map` -rule:

```
rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    shell:
        "bwa mem -R '{params.rg}' {input} "
        "| samtools view -Sb - > {output}"
```

Rule Parameters

This is usually part of the configuration file(s), too.
In your config file:

```
bwa_map:
    rg=xxx
```

and in your code:

```
rule bwa_map:
  params:
      rg=config['bwa_map']['rg']
```

Now, your workflow is configurable!

Logging

Hint

When executing a large workflow, it is usually desirable to store the logging output of each job into a separate file, instead of just printing all logging output to the terminal—when multiple jobs are run in parallel, this would result in chaotic output. - We already observed this!

Logging

Hint

When executing a large workflow, it is usually desirable to store the logging output of each job into a separate file, instead of just printing all logging output to the terminal—when multiple jobs are run in parallel, this would result in chaotic output. - We already observed this!

Task

Log into the workflow: Name all rules, which merit gathering logs.

Logging

Hint

When executing a large workflow, it is usually desirable to store the logging output of each job into a separate file, instead of just printing all logging output to the terminal—when multiple jobs are run in parallel, this would result in chaotic output. - We already observed this!

Task

Log into the workflow: Name all rules, which merit gathering logs.

Documentation

In theory *every* rules will profit from logging directives. Only in very rare cases a script or application *cannot* produce output - you will probably not be able to recognize such a case.

Logging - II

`Snakemake` offers a `log` -directive for this purpose.

```
rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    log: "logs/bwa_mem/{sample.log}"
    # we combine stdout & stdout
    shell:
        "(bwa mem -R '{params.rg}' {input} "
        "| samtools view -Sb - > {output}) 2> {log}"
```

Task

Please try this out.

Logging - III

Questions

- What did you observe? Why is there a distinction per sample?

Logging - III

Questions

What did you observe? Why is there a distinction per sample?

Documentation

This rule's output will not clutter the terminal any more. All necessary information is in a log *per* sample: If one attempt to map goes wrong, the specific error can be found in the sample log.

Temporary Files

Let's face it: Most output files of a workflow, particularly more complex ones(!), do not need to be archived. They will only take up storage until someone takes the  .
Luckily, `Snakemake` provides a way to mark a file as "temporary" and will delete it, when its not needed any longer.

Temporary Files

Let's face it: Most output files of a workflow, particularly more complex ones(!), do not need to be archived. They will only take up storage until someone takes the  .
Luckily, `Snakemake` provides a way to mark a file as "temporary" and will delete it, when its not needed any longer.

Question

The output of which rules in our workflow should be marked "temporary"?

Temporary Files - Reasoning

Rule	Temp?	Reason
all		

Temporary Files - Reasoning

Rule	Temp?	Reason
all	**No**	This rule desingates the final target(s).
bwa_map		

Temporary Files - Reasoning

Rule	Temp?	Reason
all	**No**	This rule desingates the final target(s).
bwa_map	**Yes**	Mappings are derived from the intial input and can be created reproducibly again.
samtools_sort		

Temporary Files - Reasoning

Rule	Temp?	Reason
`all`	**No**	This rule desingates the final target(s).
`bwa_map`	**Yes**	Mappings are derived from the intial input and can be created reproducibly again.
`samtools_sort`	**Yes**	Same reason as with `bwa_map`
`samtools_index`		

Temporary Files - Reasoning

Rule	Temp?	Reason
`all`	**No**	This rule desingates the final target(s).
`bwa_map`	**Yes**	Mappings are derived from the intial input and can be created reproducibly again.
`samtools_sort`	**Yes**	Same reason as with `bwa_map`
`samtools_index`	**Yes**	Same reason as with `bwa_map`
`bcftools_call`		

Temporary Files - Reasoning

Rule	Temp?	Reason
`all`	**No**	This rule desingates the final target(s).
`bwa_map`	**Yes**	Mappings are derived from the intial input and can be created reproducibly again.
`samtools_sort`	**Yes**	Same reason as with `bwa_map`
`samtools_index`	**Yes**	Same reason as with `bwa_map`
`bcftools_call`	**No**	This is a result file!
`plot_quals`		

Temporary Files - Reasoning

Rule	Temp?	Reason
`all`	**No**	This rule desingates the final target(s).
`bwa_map`	**Yes**	Mappings are derived from the intial input and can be created reproducibly again.
`samtools_sort`	**Yes**	Same reason as with `bwa_map`
`samtools_index`	**Yes**	Same reason as with `bwa_map`
`bcftools_call`	**No**	This is a result file!
`plot_quals`	**No**	This is a result file! See `all` .

Hands On XVIII – Temporary Files - Put to Work

Snakemakes way to mark outputs as temporary or intermediate is the `temp()` -function, e.g.:

```
rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        temp("mapped_reads/{sample}.bam")
    ...
```

Hands On XIX – Temporary Files - Put to Work

Snakemakes way to mark outputs as temporary or intermediate is the `temp()` -function, e. g.:

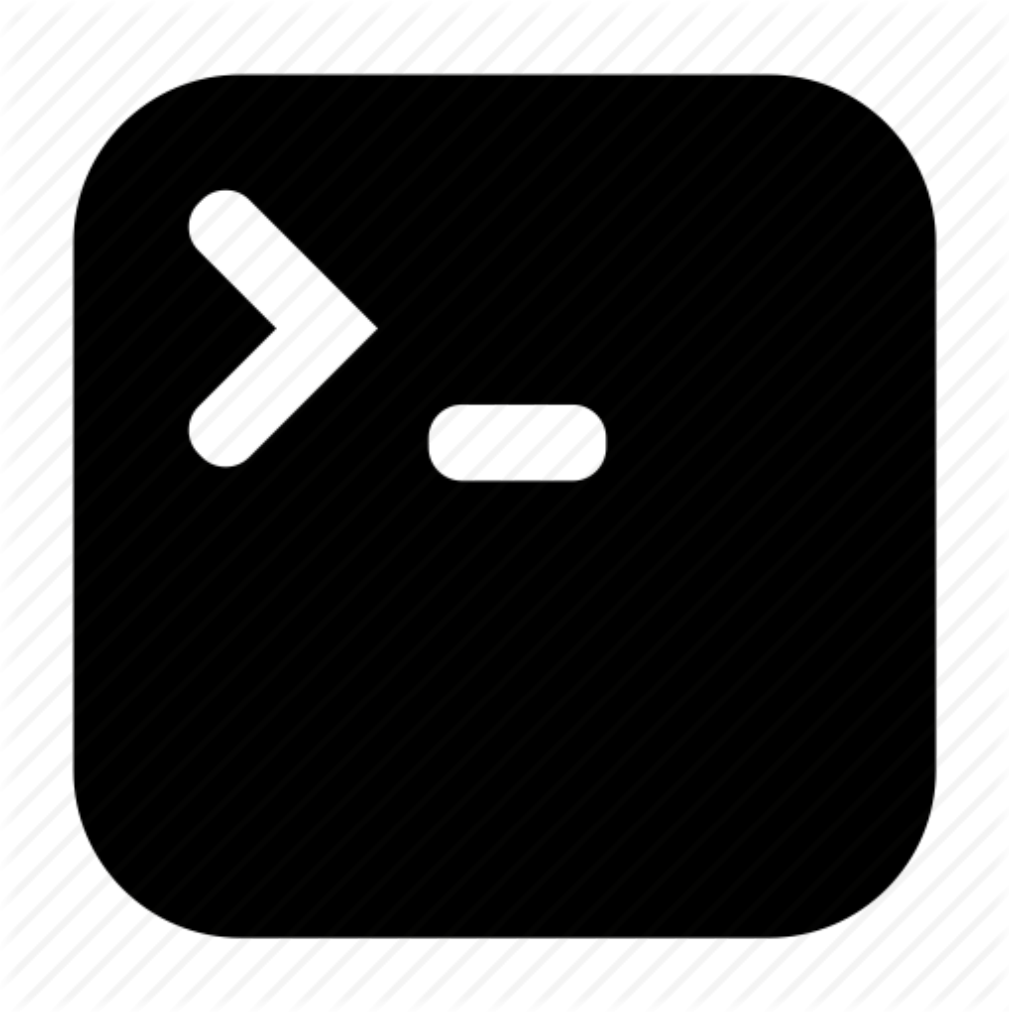
```
rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        temp("mapped_reads/{sample}.bam")
    ...
```

Please run:

```
$ du -sh *reads
```

Add the `temp()` function to `bwa_map` - and `samtools_sort` -rules, re-run the workflow and the `du` -command. What do you observer?

Parameterization via the Command Line



Executor Selection

`Snakemake` lets you select various executors. Not happy with MOGON? Take another cluster or Google Lifescience, Tibanna, Kubernetes, . . .
We may happily select the most prominent HPC batch system, the one running on MOGON, too:

```
$ snakemake --slurm
```

Now, *every* rule will submit its jobs as HPC compute jobs.

Hint

We will learn how to avoid this, soon-ish.

Default Resources for SLURM

Without specifying our SLURM-account and a (default) partition, submitting batch jobs will fail. Snakemake allows to define so-called default resources (using `--default-resources`). With them our minimal command line becomes:

```
$ snakemake --slurm \
> --default-resources slurm_account=m2_jgu-ngstraining \
>                     slurm_partition=smp
```

Hint

Please notice the missing quotation marks! All arguments belong to one parameter.

Documentation

We choose `slurm_...` -prefixes to distinguish from other non-SLURM accounts and similar stuff.

The “Beauty” of Clusters

A HPC cluster:

- offers great resources
- may hold jobs pending until resources are available!

The “Beauty” of Clusters

A HPC cluster:

- offers great resources
- may hold jobs pending until resources are available!

`Snakemake` offers a semaphore to throttle resource usage, called `--jobs/-j`. We can now write `-j unlimited` in place for `--cores 1`. Let us try

```
$ snakemake --slurm \
> --default-resources \
>    slurm_account=m2_jgu-ngstraining \
>    slurm_partition=smp
> -j unlimited
```

together. (in a sec.)

SLURM supporting Features of Snakemake

Snakemake will

- hold track of your job status (frequency of checks can be adjusted)

SLURM supporting Features of Snakemake

Snakemake will

- hold track of your job status (frequency of checks can be adjusted)
- cancel your jobs, when itself is stopped

SLURM supporting Features of Snakemake

Snakemake will

- hold track of your job status (frequency of checks can be adjusted)
- cancel your jobs, when itself is stopped
- track resource consumption (generated with `--report [FILE]`)

`SLURM` supporting Features of `Snakemake`

`Snakemake` will

- hold track of your job status (frequency of checks can be adjusted)
- cancel your jobs, when itself is stopped
- track resource consumption (generated with `--report [FILE]`)
- with `-j unlimited` we allow for an unlimited jumber of spawned jobs!

`SLURM` supporting Features of `Snakemake`

`Snakemake` will

- hold track of your job status (frequency of checks can be adjusted)
- cancel your jobs, when itself is stopped
- track resource consumption (generated with `--report [FILE]`)
- with `-j unlimited` we allow for an unlimited jumber of spawned jobs!

SLURM supporting Features of Snakemake

Snakemake will

- hold track of your job status (frequency of checks can be adjusted)
- cancel your jobs, when itself is stopped
- track resource consumption (generated with `--report [FILE]`)
- with `-j unlimited` we allow for an unlimited jumber of spawned jobs!

Warning

Unlimited number of jobs may yield in I/O contention and too many calls to check the job status. Use with care for both issues there is a remedy, which we will meet later!

Hands On XX – Running the Worklow as Cluster Jobs

Our workflow is tiny. It does not merit cluster execution. Yet, for the purpose of this course, please run (with `-F` to *enforce* execution):

```
$ snakemake --slurm \
> --default-resources slurm_account=m2_jgu-ngstraining \
>                     slurm_partition=smp \
> -j unlimited -F
```

Question

- Observations?

The Downside of working with Defaults

Ok, this should(!) be running, yet you saw the warnings.

Question

Why will a “real” workflow fail with all these default settings?

The Downside of working with Defaults

Ok, this should(!) be running, yet you saw the warnings.

Question

Why will a “real” workflow fail with all these default settings?

- “real data” tend to be biffer - insufficient memory
- “real runs” tend to take longer - insufficient wall time
- “real runs” tend to run into I/O issues - insufficient workflows

Cluster Configuration

Hint

Best abstract the configuration for a cluster from the workflow!

In addition to a `config` directory to hold our samples, `Snakemake` will accept a directory with profile settings, usually “ `profile` ”.

Cluster Configuration

Hint

Best abstract the configuration for a cluster from the workflow!

In addition to a `config` directory to hold our samples, `Snakemake` will accept a directory with profile settings, usually "`profile`". A profile has to adhere to `YAML`-format and be named `config.yaml`. An example for a cluster configuration might look like (sorry, no syntax highlighting available):

```
set-resources:
    bwa_map:
        runtime: 5
        mem_mb_per_cpu: 1800
```

Cluster Configuration

Hint

Best abstract the configuration for a cluster from the workflow!

In addition to a `config` directory to hold our samples, `Snakemake` will accept a directory with profile settings, usually "`profile`". A profile has to adhere to `YAML`-format and be named `config.yaml`. An example for a cluster configuration might look like (sorry, no syntax highlighting available):

```
set-resources:
    bwa_map:
        runtime: 5
        mem_mb_per_cpu: 1800
```

`Snakemake` offers many more cluster configuration details, we will revisit some during this course.

Hands On XXI – Cluster Execution **with** Configuration

Please execute the workflow again, now with

```
$ # first copy the template profile:
$ cp -r ../profile .
$ snakemake --slurm \
> --default-resources slurm_account=m2_jgu-ngstraining \
>                     slurm_partition=smp \
> -j unlimited --workflow-profile ./profile/ -F
```

Question

What do you observe? Which rule still provokes a warning?

Local Rules

Some rules do *not* merit cluster execution!

Local Rules

Some rules do *not* merit cluster execution!

As a rule of thumb this (almost) always includes

1. plotting rules
2. rules which perform downloads
3. rules which include remote files (e. g. to an archive)

Local Rules

Some rules do *not* merit cluster execution!

As a rule of thumb this (almost) always includes

1. plotting rules
2. rules which perform downloads
3. rules which include remote files (e. g. to an archive)

For such a purpose `Snakemake` offer the `localrules` directive.

```
# its useage is (at or near the top of a Snakefile)
localrules: rule_a, rule_b, rule_c, ...
```

Hands On XXII – Implementing "`localrules`"

Please add

```
localrules: plot_quals
```

at the top of your `Snakefile` and rerun the workflow on this cluster.

Hands On XXIII – Implementing "`localrules`"

Please add

```
localrules: plot_quals
```

at the top of your `Snakefile` and rerun the workflow on this cluster.

Question

Which is the impact on the overall execution time?

Hands On XXIV – Implementing "`localrules`"

Please add

```
localrules: plot_quals
```

at the top of your `Snakefile` and rerun the workflow on this cluster.

Question

Which is the impact on the overall execution time?

Hint

Rules which do are not compute intensive are best executed locally. The overhead of submitting a job and waiting for it to finish will not be an issue.

Outline

What is this about?

Questions

- Is there a way to get benchmarking information?
- When I want to write a publication, which software(s) version(s) have I been using?

Objectives

1. Learn the implications and limitations of benchmarks.
2. Learning how to generate and interprate reports with `Snakemake`.

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality
- assessing software completeness (all the features we want?)

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality
- assessing software completeness (all the features we want?)
- assessing software usability

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality
- assessing software completeness (all the features we want?)
- assessing software usability
- . . .

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality
- assessing software completeness (all the features we want?)
- assessing software usability
- . . .

What is “Benchmarking” in Computer Science?

Benchmarking may mean a number of things:

- assessing software quality
- assessing software completeness (all the features we want?)
- assessing software usability
- …

Documentation

What is most commonly meant: Getting performance and scalability markers.

Generating Benchmarks

Similar to crating a logfile, we are provided with a `benchmark` -directive, which can point to a bechmarking file:

```
rule bwa_map:
    ...
    log:
        "logs/bwa_mem/{sample}.log"
    benchmark:
        "benchmarks/{sample}.bwa.benchmark.txt"
    ...
```

Documentation

`Snakemake` will measure the wall clock time and memory usage (in MiB) and store it in the file in tab-delimited format.

Repeating Benchmarks

Repeated benchmarks

It is possible to repeat a benchmark multiple times in order to get a sense for the variability of the measurements. This can be done by annotating the benchmark file, e. g., with `repeat("benchmarks/{sample}.bwa.benchmark.txt", 3)`. `Snakemake` can be told to run the job three times. The repeated measurements occur as subsequent lines in the tab-delimited benchmark file.

“Proper” Benchmarking

A good estimate of a software’s scalability takes into account:

- running with different degrees of parallelization, e. g. 1-n threads

“Proper” Benchmarking

A good estimate of a software’s scalability takes into account:

- running with different degrees of parallelization, e. g. 1-n threads
- not using toy inputs, but real data, possibly different real data

“Proper” Benchmarking

A good estimate of a software’s scalability takes into account:

- running with different degrees of parallelization, e. g. 1-n threads
- not using toy inputs, but real data, possibly different real data
- avoiding I/O issues (common with alignment programs)

“Proper” Benchmarking

A good estimate of a software’s scalability takes into account:

- running with different degrees of parallelization, e. g. 1-n threads
- not using toy inputs, but real data, possibly different real data
- avoiding I/O issues (common with alignment programs)

“Proper” Benchmarking

A good estimate of a software’s scalability takes into account:

- running with different degrees of parallelization, e. g. 1-n threads
- not using toy inputs, but real data, possibly different real data
- avoiding I/O issues (common with alignment programs)

Warning

We conclude: `Snakemake`’s benchmarking capabilities are limited!

Snakemake Reports

Asking Snakemake for a report on a completed workflow is as easy as:

```
$ snakemake --report
```

This will generate a file called "`report.html`", which you can visualize with a browser.

Snakemake Reports

Asking Snakemake for a report on a completed workflow is as easy as:

```
$ snakemake --report
```

This will generate a file called "report.html", which you can visualize with a browser.

Viewing the Report

Due to a bug on MOGON: Copy the file to your computer and visualize it in your browser!

Snakemake Reports and Cluster Jobs

Why reports about Cluster Jobs can be misleading.

When reporting about a cluster or cloud job, Snakemake sheperd job on the login-node will measure the wall time from submit time to the finish time, not the executing job(s).
This means: Times can be greatly exegerated!

Outline

What is this about?

Questions

- How do I get a workflow for a given scientific problem?
- How do I run such an arbitrary workflow?

Objectives

1. Introducing the worfkflow catalogue?
2. Learning the difference between “curation” (what some people think) and “curation” (what really works)?

Selecting and Downloading from the Workflow Catalogue

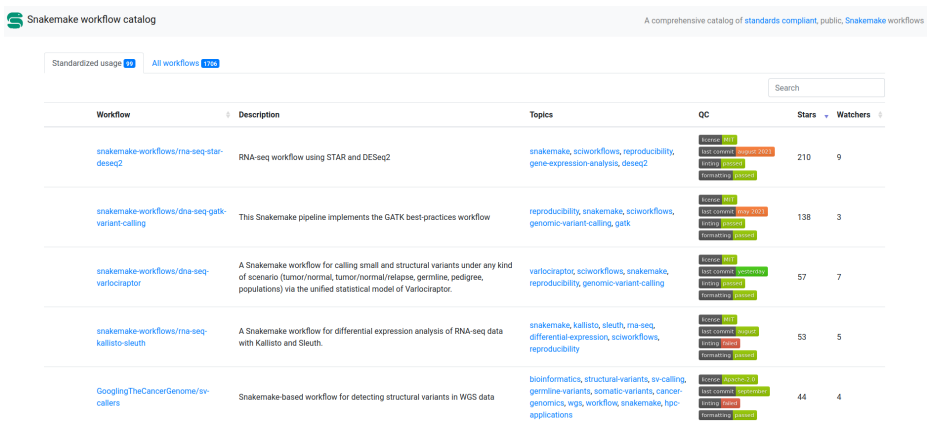
You can find the Snakemake worfkflow catalogue, here. It makes a difference between workflows which meet best-practice criteria - and those which do not.

You can download and run any workflow.

Selecting and Downloading from the Workflow Catalogue

You can find the Snakemake worfkflow catalogue, here. It makes a difference between workflows which meet best-practice criteria - and those which do not.

You can download and run any workflow.



Warning

Except, you most likely cannot, because of a missing cluster configuration and some missing features.

Deployment

Select (=click on) any desired workflow. There are two alternatives:

1. a workflow offers a release - in which case you can download and unpack it
2. all workflows offers a “ `git clone` ” hint

Running Workflows on Cluster (or other environment)

Most likely a specific workflow never has been testing on *your* computer before. It is almost ensured it will run on arbitrary servers, but clusters are a different story.
So

- try to parameterize your workflow as learned

Running Workflows on Cluster (or other environment)

Most likely a specific workflow never has been testing on *your* computer before. It is almost ensured it will run on arbitrary servers, but clusters are a different story.
So

- try to parameterize your workflow as learned
- if it gives issues and you know how to correct it, “fork” the worklow and create a pull request

Running Workflows on Cluster (or other environment)

Most likely a specific workflow never has been testing on *your* computer before. It is almost ensured it will run on arbitrary servers, but clusters are a different story.
So

- try to parameterize your workflow as learned
- if it gives issues and you know how to correct it, “fork” the worklow and create a pull request
- if you cannot fix it, create a bug report

Interlude – Learn git!

If you do not know what “fork” and “pull request” means, learn git!

- there are courses

Interlude – Learn git!

If you do not know what “fork” and “pull request” means, learn git!

- there are courses
- and lots of online material

Interlude – Learn git!

If you do not know what “fork” and “pull request” means, learn git!

- there are courses
- and lots of online material
- and books

Interlude – Learn git!

If you do not know what “fork” and “pull request” means, learn git!

- there are courses
- and lots of online material
- and books

Interlude – Learn git!

If you do not know what “fork” and “pull request” means, learn git!

- there are courses
- and lots of online material
- and books

Warning

Knowing git is essential in data analysis!

Outline

What is this about?

Questions

- How do we add execution parameters?
- How do we tune scientific parameters?

Objectives

1. Learn to use parameters relevant for the batch systems.
2. Learn how to tune `Snakemake` on the command line.
3. Learn how to tune `Snakemake` with configuration files.