



# snakemake

A framework for reproducible data analysis

>7 new citations per week

>370,000 downloads

Open source, MIT licensed

READ THE ROLLING PAPER

READ THE DOCS

WATCH THE INTRO

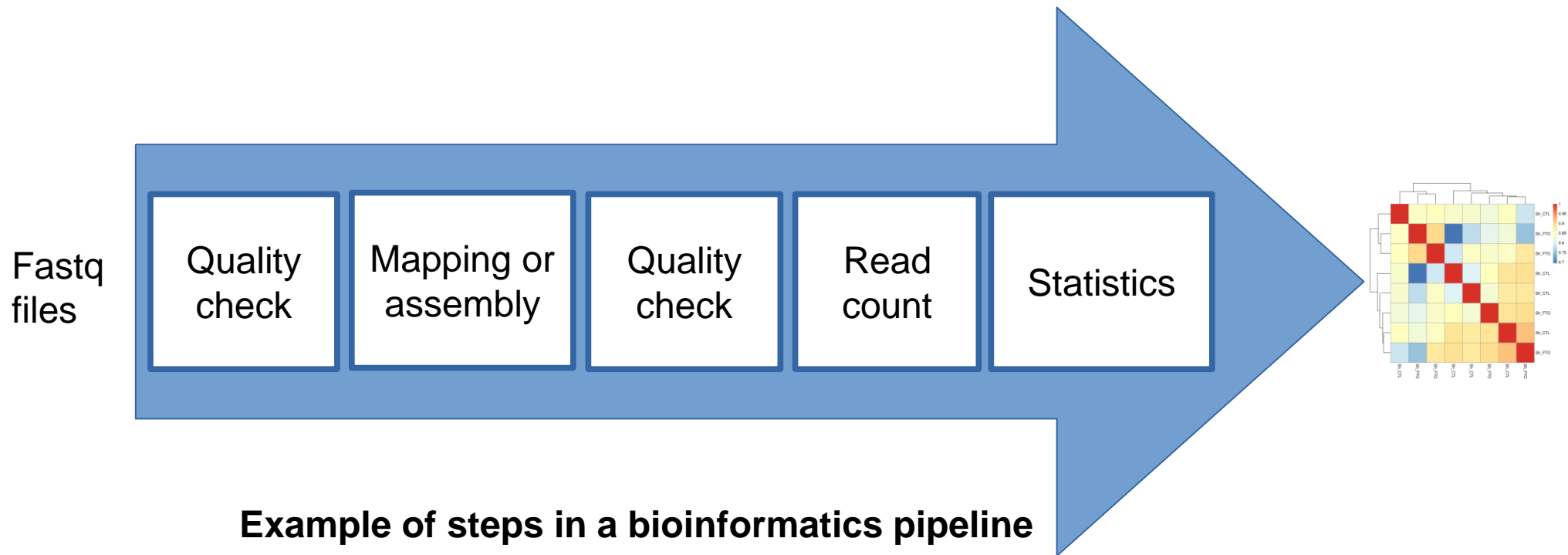
VIEW ON GITHUB

FOLLOW ON TWITTER

<https://snakemake.github.io/>

# Part I – Bioinformatic pipeline & snakemake

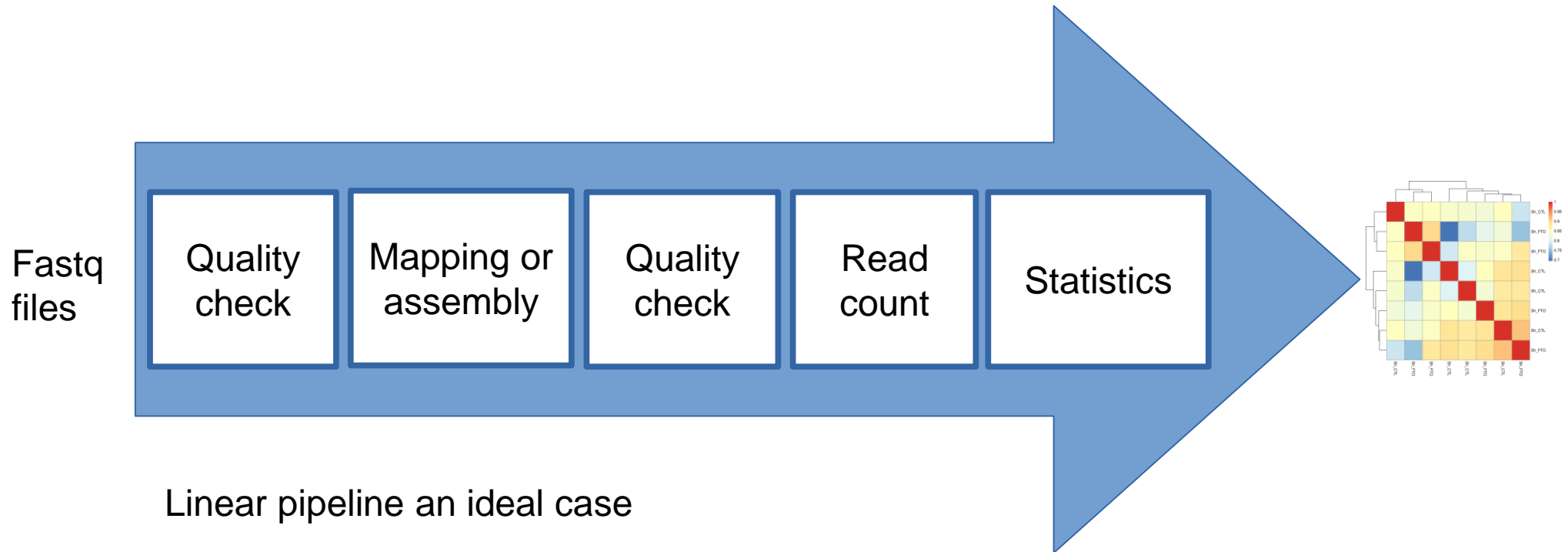
# Bioinformatic analyses pipeline



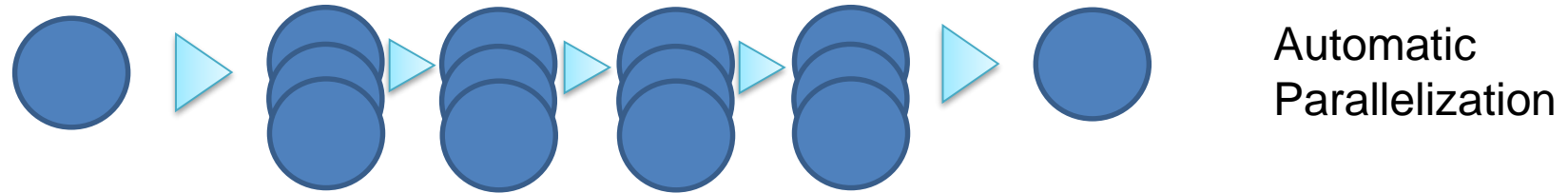
## Why is a workflow manager needed?

- automation → multiple input files
- reproducibility → externalization of parameters, version of tools
- standardisation → error reduction
- execution optimisation → redo only the missing part in case of unplanned machine stop

# Advantages of using snakemake

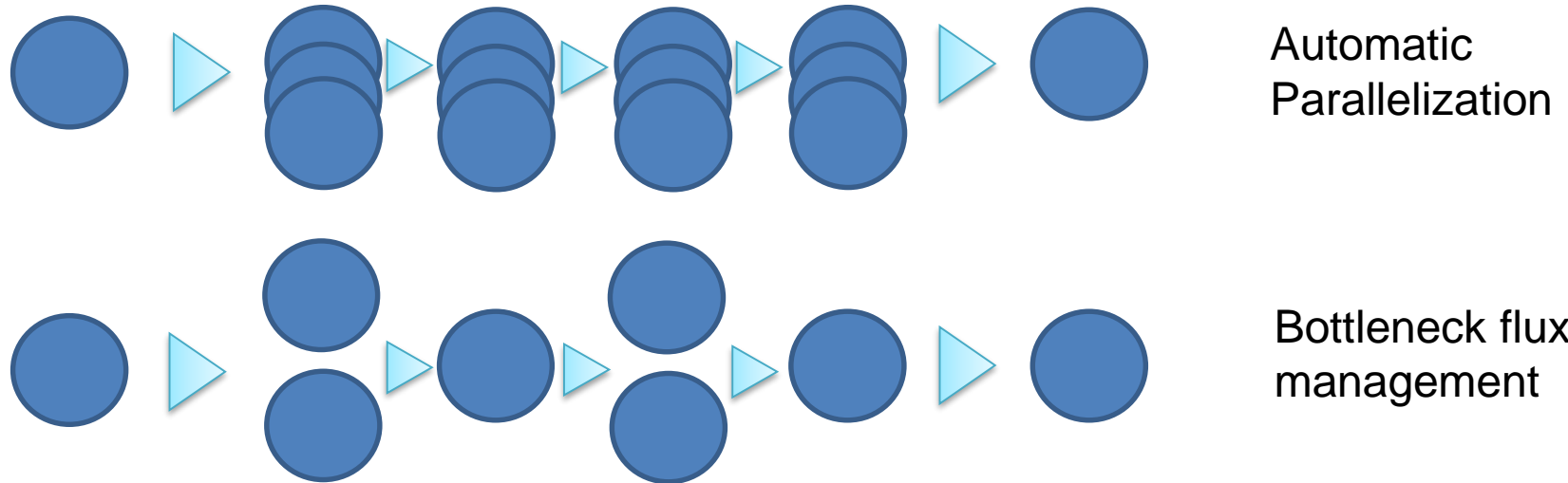


# Advantages of using snakemake



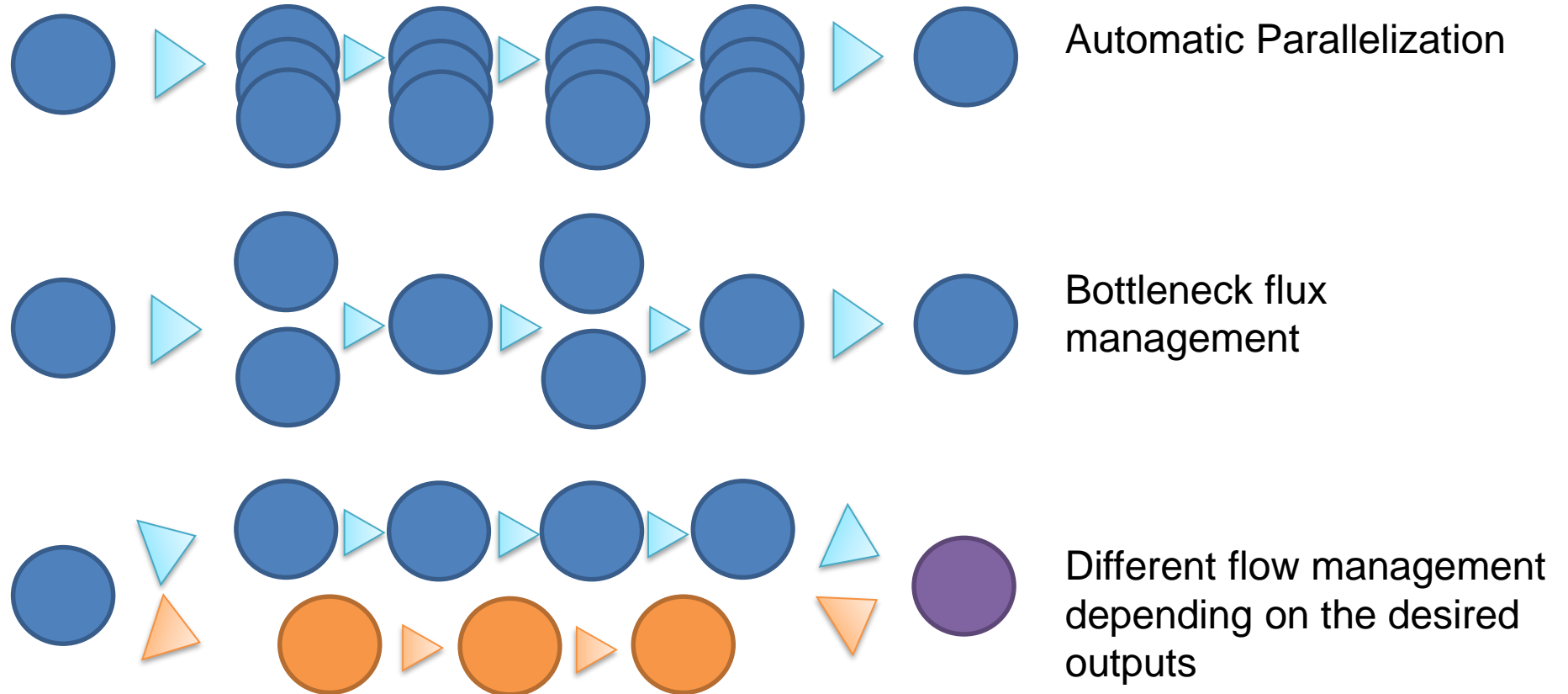
Each step is repeated according to the number of samples

# Advantages of using snakemake

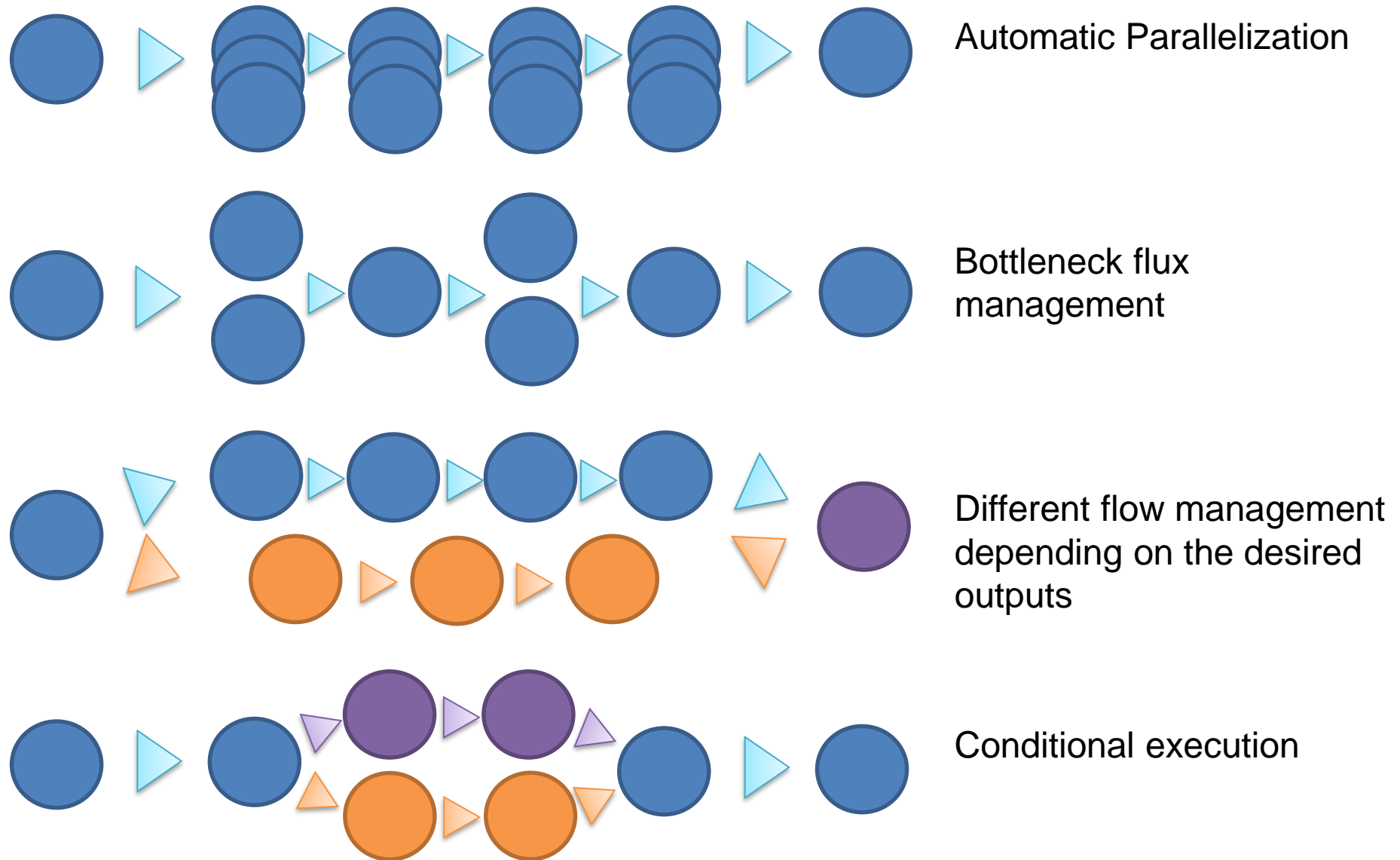


Some steps reduce the number of inputs to a single output file and others split a single file into several outputs

# Advantages of using snakemake



# Advantages of using snakemake

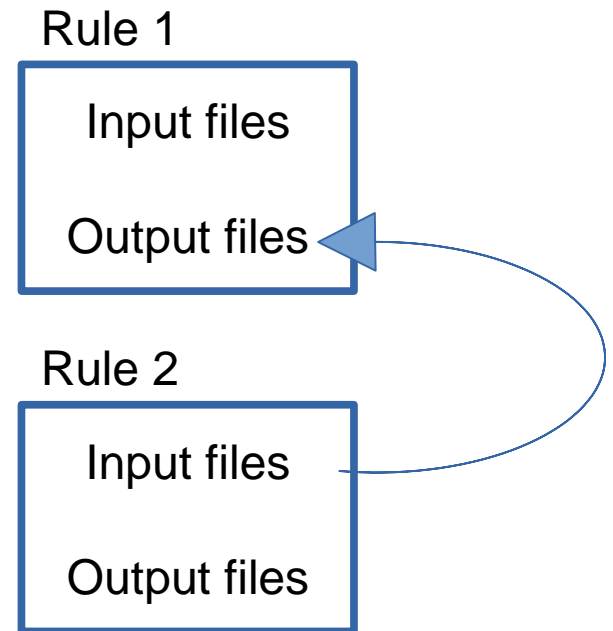




# Snakemake philosophy

**The central idea of Snakemake is that workflows are specified through decomposition into steps represented as rules.**

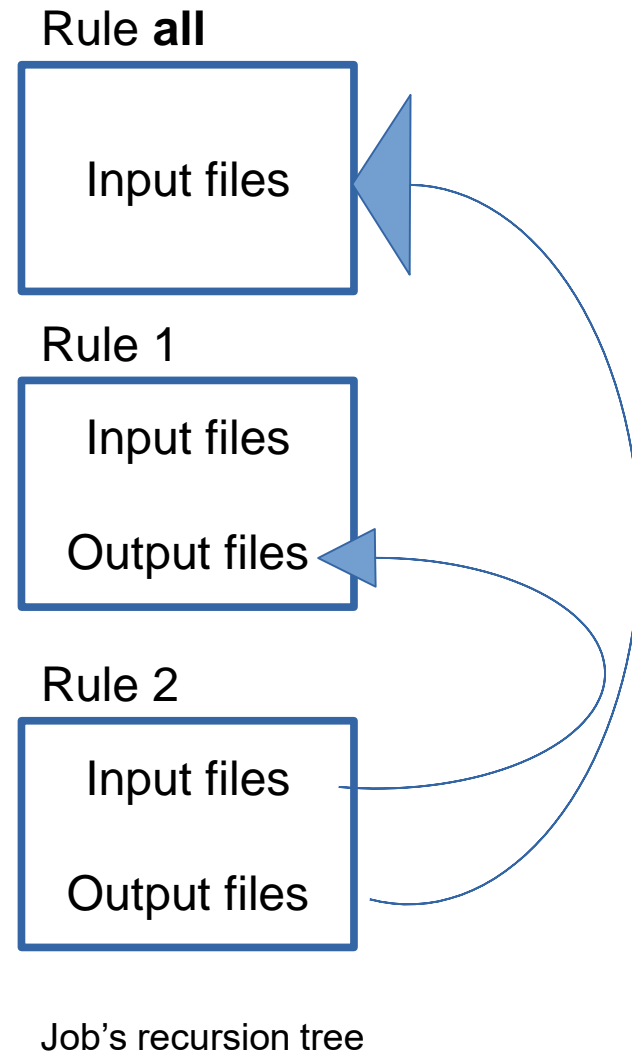
Each rule describes how to obtain a set of output files from a set of input files.



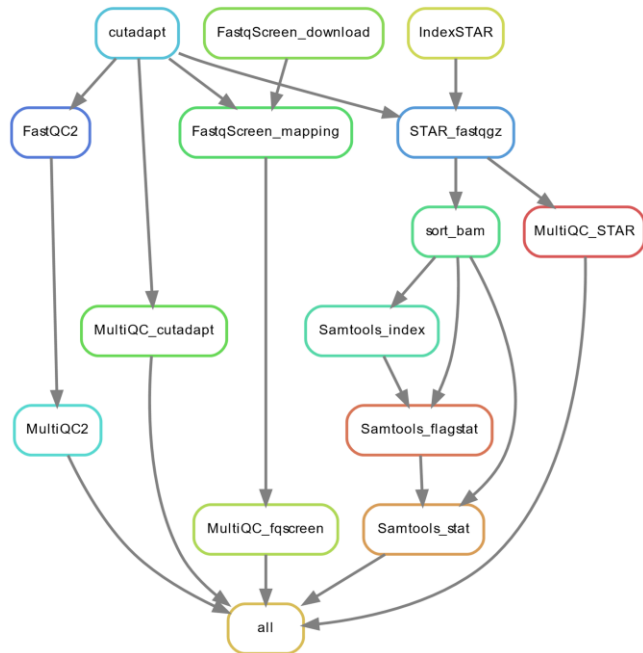
# Snakemake philosophy

Snakemake **goes on recursively** for the latter, **until all input files of all jobs are either generated by another job or already present in the used storage** (e.g., on disk).

The files required at the end of the pipeline are given in a specific rule called « all » or « defaults ».

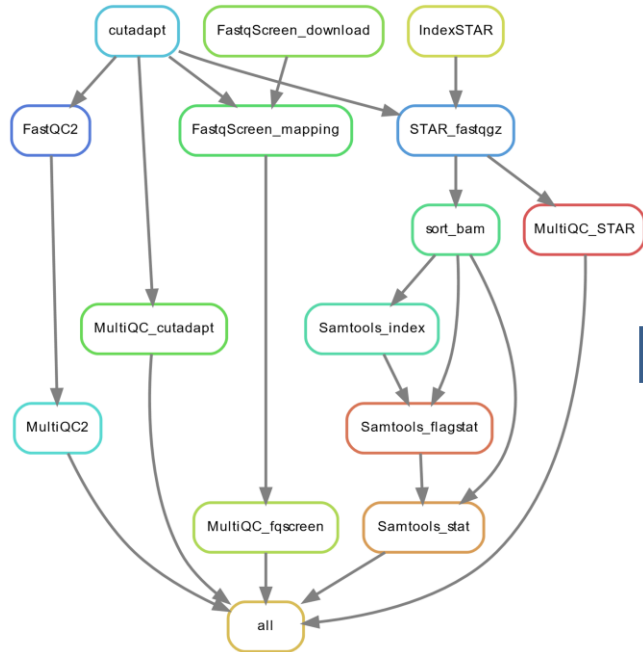


# Directed acyclic graph (DAG)

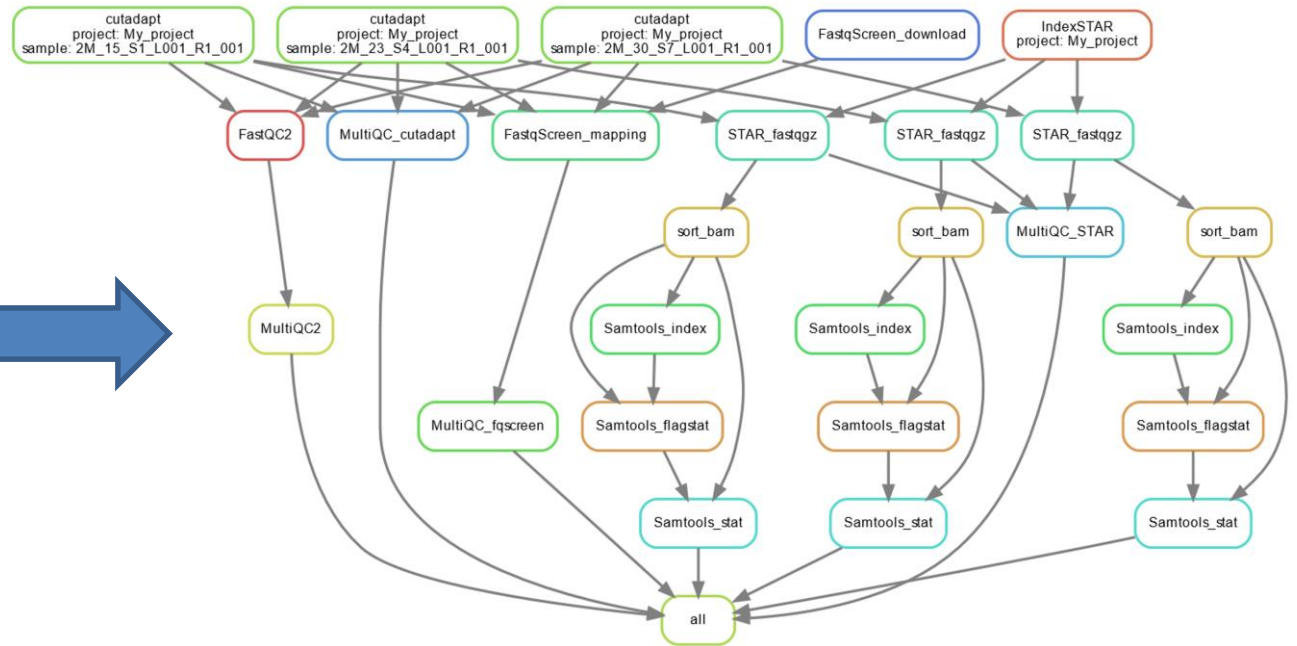


Rule graph

# Directed acyclic graph (DAG)



Rule graph

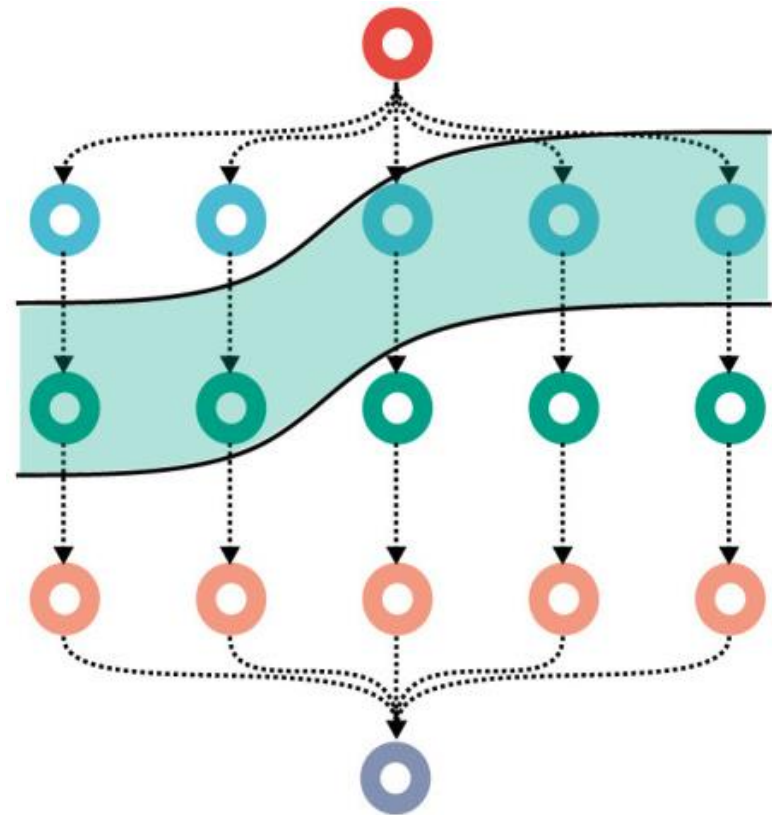


Job graph

# Job scheduling

Because of their dependencies, not all jobs (rules) in a workflow can be executed at the same time.

Snakemake solves the scheduling problem at the beginning of the workflow execution and whenever a job has finished and new jobs become pending.



Part II – Be reproducible

# Package manager

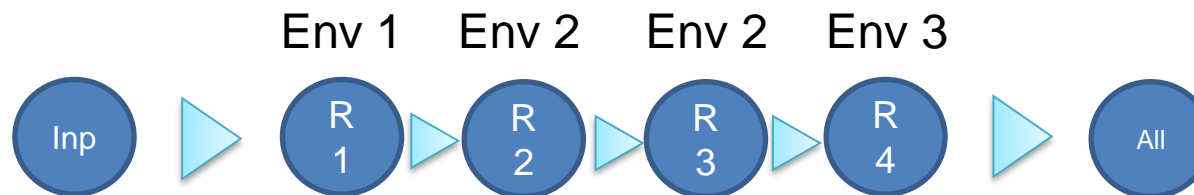


**Snakemake** allows each rule to define a **software environment** that will be **automatically deployed via the Conda** package manager.

Almost all **bioinformatics tools** are available from the **anaconda** package repository: <https://anaconda.org/anaconda/repo>

For a rule, Snakemake will simply activate the given **environment**, instead of automatically deploying anything.

Alternatively to Conda, **containers** can be used (**Docker or Singularity**)



# Configuration

Snakemake can use a **configuration file** (ex: Config.yaml).

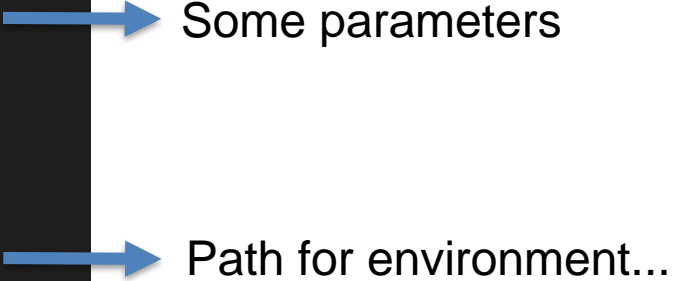
This file **should contain all the specific** of a project and externalize **constant** or **parameters**.

It can be **defined directly in the Snakefile or on the command line** at runtime.

You can have different configuration files in your workflow.

```
# Rules threads
threads:
  low: 4
  medium: 8
  high: 16

# Conda environment
env:
  QCF: "envs/QCF.yaml"
  Mapping: "envs/Mapping.yaml"
  Rstat: "envs/Rstat.yaml"
```



Example of config file



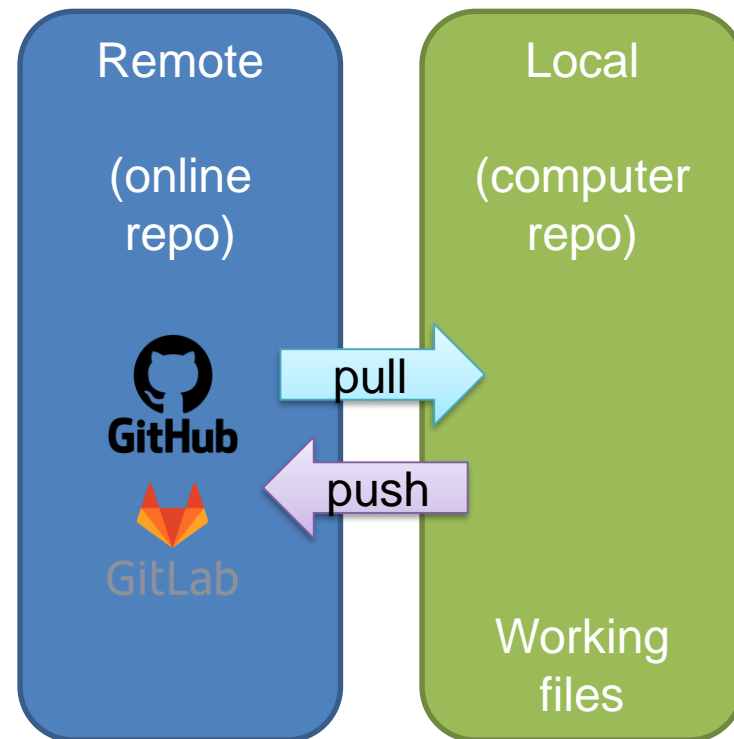
# Version control

A git repository allows you to save the progress of your work and to facilitate collaborative work on a project.

Gitlab or Github are web-based project management applications.



<https://gite.lirmm.fr/>



# Part III – What does a **snakefile** contain ?

# Rules

Rule name


```
rule sort:
  input:
    "path/to/dataset.txt"
  output:
    "dataset.sorted.txt"
  shell:
    "sort {input} > {output}"
```

How to create  
output from input

Refer to input and output  
from shell command

# Wildcards

Generalize rules with  
named wildcards



```
rule sort:
  input:
    "path/to/{dataset}.txt"
  output:
    "{dataset}.sorted.txt"
  shell:
    "sort {input} > {output}"
```

Wildcards can be used to generalize a rule allow to be applicable to a number of e.g. datasets.

Importantly, the **wildcard** names in input and output must be named identically.

**Multiple wildcards in one filename can cause ambiguity.**

# Conda environment

```
rule sort:
  input:
    "path/to/{dataset}.txt"
  output:
    "{dataset}.sorted.txt"
  conda:
    "envs/myprogram.yaml"
  shell:
    "myprogram {input} > {output}"
```

**Each environment is described by a YAML (configuration) file used by conda to install constituent software.**

**The environment is stored by snakemake and reloaded as necessary** without re-installation, unless the .yaml file has been modified.

```
name: Rstat
channels:
  - conda-forge
  - bioconda
  - anaconda
  - defaults
dependencies:
  - bioconductor-annotationdbi=1.52.0
  - bioconductor-deseq2=1.30.1
  - r-base=4.0.5
  - r-matrix=1.3_4
  - readline=8.1
  - pip=22.1.2
  - pip:
    - pandas=1.4.4
```

→ Name of the environment

→ Channels for downloading programs

→ Wanted programs and their versions

→ Specific case with pip installation for python packages

# Script

```
rule sort:
    input:
        "path/to/{dataset}.txt"
    output:
        "{dataset}.sorted.txt"
    conda:
        "envs/myprogram.yaml"
    script:
        "scripts/mytask.py"
```

Rules can be executed with different tags:

- **shell**: for a shell command
- **script**: for a python, R, markdown, Julia or Rust script
- **notebook**: for jupyter notebooks (<https://jupyter.org/try>)
- **wrappers**

When using script integration instead of shell commands, Snakemake **automatically** inserts an object **giving access to all the properties** of the job.

```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_table(snakemake.input[0])

data.hist(bins=snakemake.config["hist-bins"])

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

Import of modules

Input from snakemake rule

Some params declared in the configuration file

Output from snakemake rule

Python code in mytask.py

# Wrappers

```
rule sort:
    input:
        "path/to/{dataset}.txt"
    output:
        "{dataset}.sorted.txt"
    conda:
        "envs/myprogram.yaml"
    wrapper:
        "0.24.0/bio/mytool"
```

A wrapper is a script designed for a specific tool.

The Snakemake Wrapper Repository is a **collection of reusable wrappers** that allow you to **quickly use popular tools** from Snakemake rules and workflows.

Snakemake will **automatically download and use the corresponding wrapper** files from:

<https://github.com/snakemake/snakemake-wrappers/tree/master/bio>

Alternatively, the wrapper directive **can also point to full URLs, including the local file://**

```
rule sort:
    input:
        "path/to/{dataset}.txt"
    output:
        "{dataset}.sorted.txt"
    conda:
        "envs/myprogram.yaml"
    threads:
        8
    wrapper:
        "https://github.com/snakemake/
        snakemake-wrappers/raw/0.2.0/bio/
        samtools/sort"
```

 New tag: declare the number of threads to use in a rule

<https://snakemake-wrappers.readthedocs.io/en/stable/>

To go further



# Main tags

```
1 rule cutadapt:
2     ....# Aim: removes adapter sequences from high-throughput sequencing reads
3     ....# Use: cutadapt -a ADAPTER [options] [-o output.forward] [-p output.reverse]
4     ....# <input.forward> <input.reverse>
5     ....message:
6     ....    "cutadapt --- remove poly-A and adapters--- on {wildcards.name}"
7     ....priority: 0 # default value 0
8     ....conda: ENVIRONMENT # can be container or conda
9     ....threads: 10 # depend on your computer or cluster
10    ....resources: mem_mb=100 # depend on your computer or cluster
11    ....log:
12    ....    PATH+"out/{project}/cutadapt/{name}.log"
13    ....params:
14    ....    quality = QUALITY,
15    ....    length = LENGTH,
16    ....    cutbase = CUTBASE,
17    ....    adapters = ADAPTERS
18    ....input:
19    ....    PATH+INPFOL+"{name}.fastq.gz"
20    ....output:
21    ....    PATH+"out/{project}/cutadapt/{name}.fastq.gz"
22    ....shell:
23    ....    "cutadapt "
24    ....    "-a 'A{{14}}' " # cut poly-A
25    ....    "-a 'file:{params.adapters}' "
26    ....    "--no-indels "
27    ....    "--overlap=5 "
28    ....    "-u {params.cutbase} " # cut n first bases
29    ....    "-q {params.quality} " # filter on quality threshold
30    ....    "-m {params.length} " # keep only read with minimal length defined and >
31    ....    "-o {output} "
32    ....    "{input} "
33    ....    "1> {log}"
```

Comments your rule and appears during the execution

Rule priority in the DAG

Threads and resources define the limits of execution in terms of memory usage and parallelized flow

Log file to store the printed lines in the shell

Parameters specific to the rule which can be defined in the configuration file for generalization of the rule

Example of rule with several snakemake tags

# Architecture

```
├── .gitignore
├── README.md
├── LICENSE.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── .tests
│   ├── integration
│   └── unit
├── images
│   └── rulegraph.svg
├── workflow
│   ├── rules
│   │   ├── module1.smk
│   │   └── module2.smk
│   ├── envs
│   │   ├── tool1.yaml
│   │   └── tool2.yaml
│   ├── scripts
│   │   ├── script1.py
│   │   └── script2.R
│   ├── notebooks
│   │   ├── notebook1.py.ipynb
│   │   └── notebook2.r.ipynb
│   ├── report
│   │   ├── plot1.rst
│   │   └── plot2.rst
│   ├── Snakefile
│   └── documentation.md
├── config
│   ├── config.yaml
│   └── some-sheet.tsv
├── results
└── resources
```

**Snakemake advises to build your repo as is with :**

- **.gitignore** → write all large files that should not be stored on git
- **README.md** → all informations required to reuse your workflow
- **License** → the appropriate licence to protect your work
- **Code of conduct and Contributing** → if your project is open to community contributions; define your rules
- **.tests** → for unit tests
- **Images** → the DAG of your workflow
- **Workflow** → all the scripts and snakefiles required for the pipeline with the documentation file
- **Config** → your config files
- **Results** → results of your pipeline
- **Resources** → all files required for the analysis, e.g. reference genome/transcriptome or annotation files

# Unit tests

- In computer programming, unit testing is a software testing method by which individual units of source code sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine whether they are fit for use. ([https://en.wikipedia.org/wiki/Unit\\_testing](https://en.wikipedia.org/wiki/Unit_testing))
- **Snakemake can automatically generate a suite of unit tests** from a workflow that has already been successfully executed.
- **Each unit test consists of the execution of one rule**, using input data taken from the source workflow.
- The generated results are by default compared byte-by-byte against the results given by in the source workflow.

# Unit tests

- Importantly, note that such **unit tests shall not be generated from big data**, as they should usually be finished in a few seconds.
- The **small dummy datasets** can in addition be used to generate an integration test, that could e.g. be stored under `.tests/integration`, next to the unit tests.
- Each auto-generated unit test is stored in a file `.tests/unit/test_<rulename>.py`, and executes just the one representative job of the respective rule.

```
snakemake -s Snakefile --generate-unit-tests
```