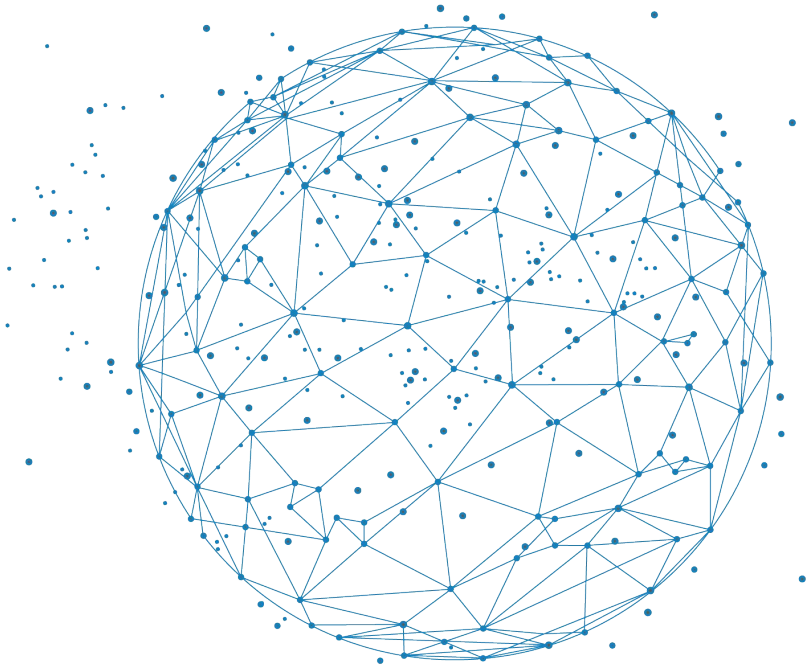
FAIR Bioinfo 2024 - Strasbourg



snakemake

Introduction to snakemake workflows

Valérie Cognat

Slides inspired from C. Toffano-nioche (I2BC) [IFB]

Johannes Koester [Snakemake tutorial]

BiGEst IFB INSTITUT FRANÇAIS DE BIOINFORMATIQUE elixir FRANCE cnrs INRAE Inserm cea INVESTIR L'AVENIR

Introduction to snakemake workflows

- What is a **Workflow** ?
- The **snakemake rule** concept
- The **snakefile**
- First steps with snakemake
- The best practices

What is a workflow ?

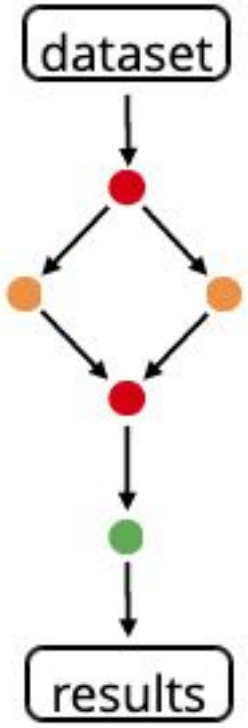
IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea INVESTIR L'AVENIR

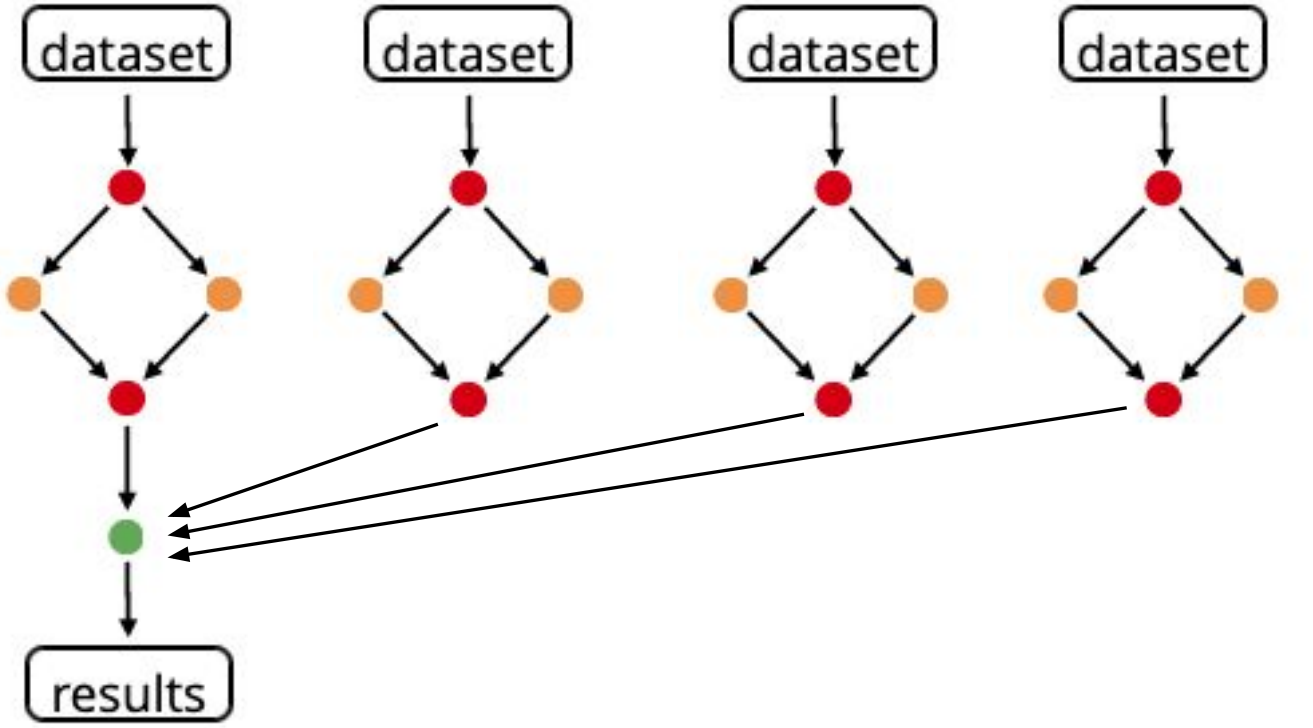
Workflow definition

- a **pool of commands**, progressively linked by the treatments, from the input data towards the results



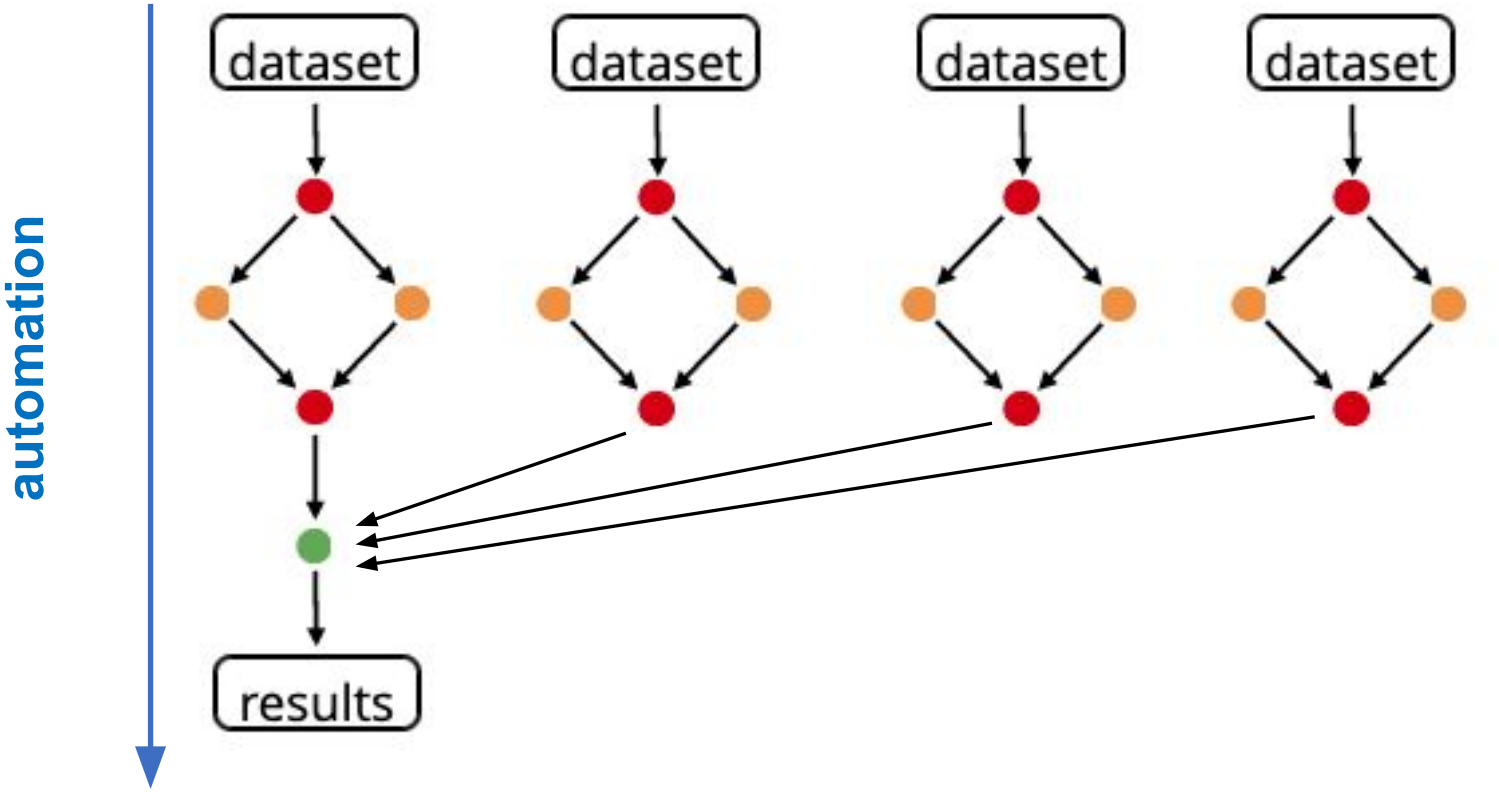
Workflow definition

- a **pool of commands**, progressively linked by the treatments, from the input data towards the results

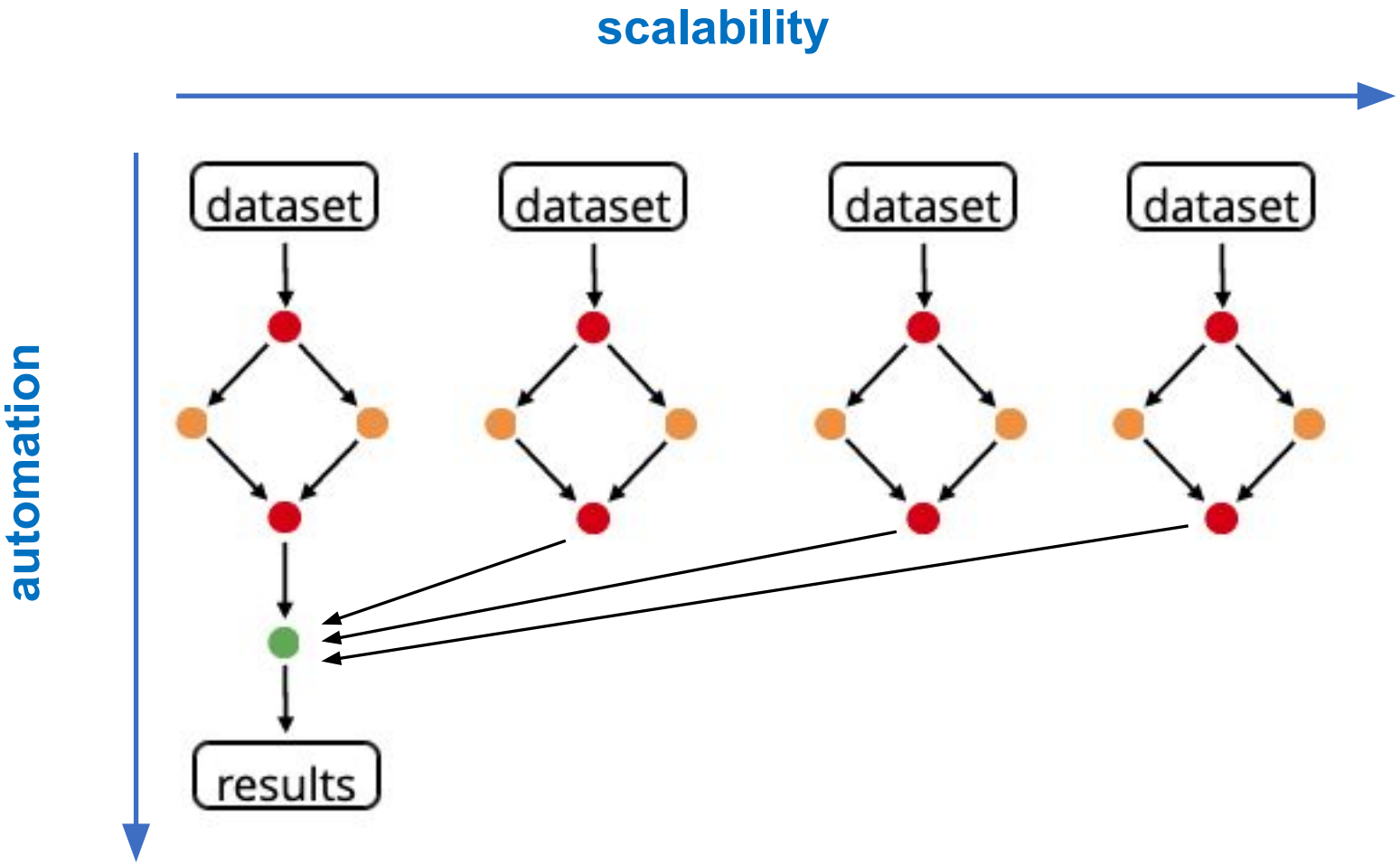


- In case of **data parallelization**, several data flows can be processed in parallel

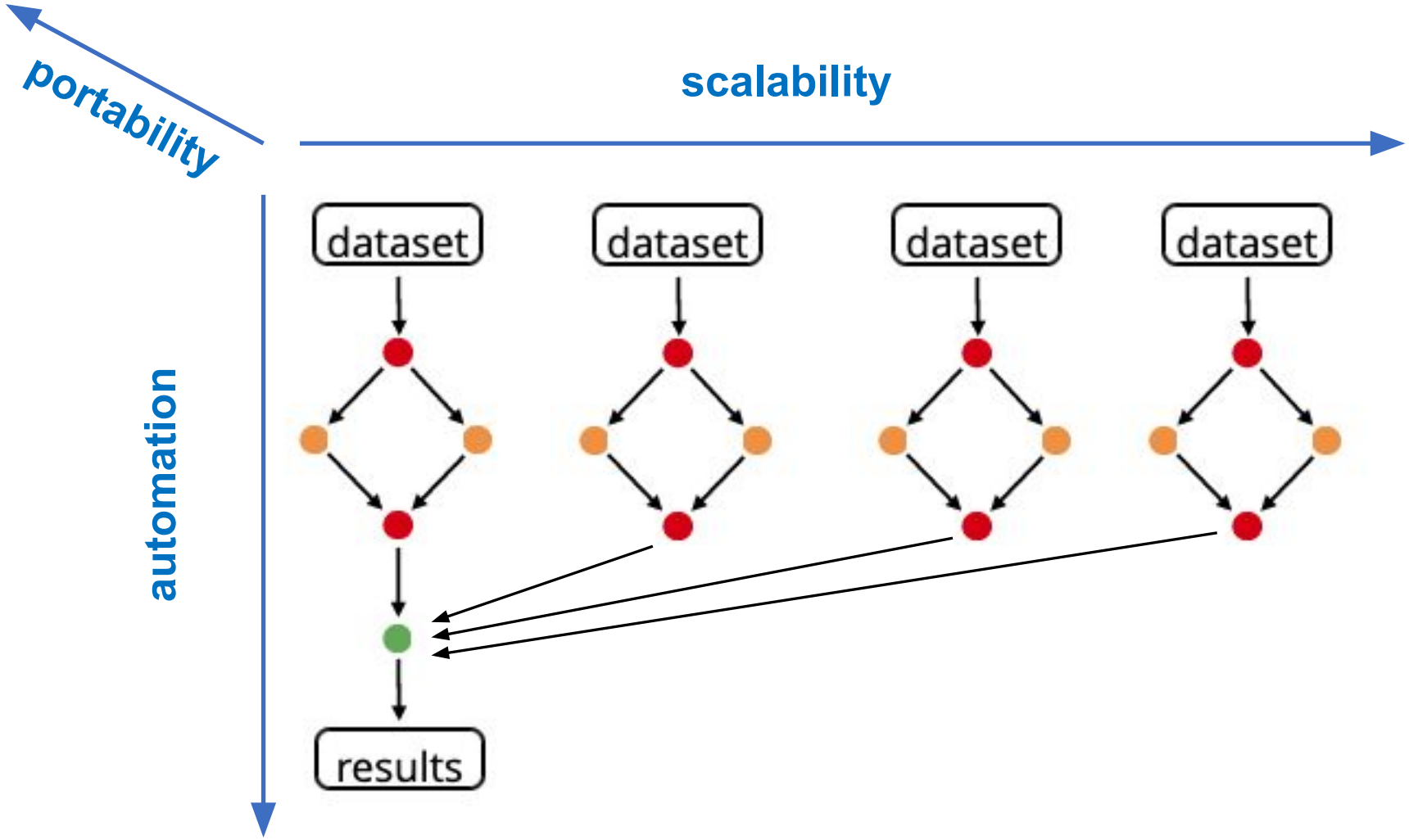
Workflow definition



Workflow definition



Workflow definition



Workflow management system

- **Many workflow management systems**, many forms:
  - Command line: shell (need to script parallelization alone, not easy)
  - Rule: CMake, snakemake, nextflow
  - Graphic interface: Galaxy, Taverna, Keppler, …
- Pros:
  - Reproducibility: keep track (when files were generated & how)
  - Step dependency
  - Manage parallelization (error recovery)
- Cons:
  - Learning effort

snakemake

IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea INVESTIR L'AVENIR

History

BIOINFORMATICS APPLICATION NOTE

Vol. 28 no. 19 2012, pages 2520–2522
doi:10.1093/bioinformatics/bts480

Genome analysis

Advance Access publication August 20, 2012

Snakemake—a scalable bioinformatics workflow engine

Johannes Köster[1,2,*] and Sven Rahmann[1]

[1]Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen and [2]Paediatric Oncology, University Childrens Hospital, 45147 Essen, Germany

Associate Editor: Alfonso Valencia

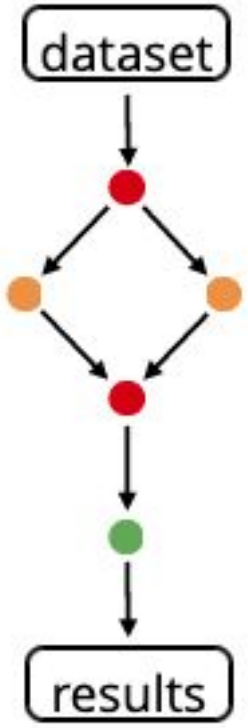
- 2012
- Johannes Köster
- Open source, MIT license
- Mix of the programming language **Python** (snake) and the rule-based automation tool **Make**
- Currently : version ~~8.9.0~~ ~~8.10.0~~ 8.10.6 (python 3.11 & 3.12)

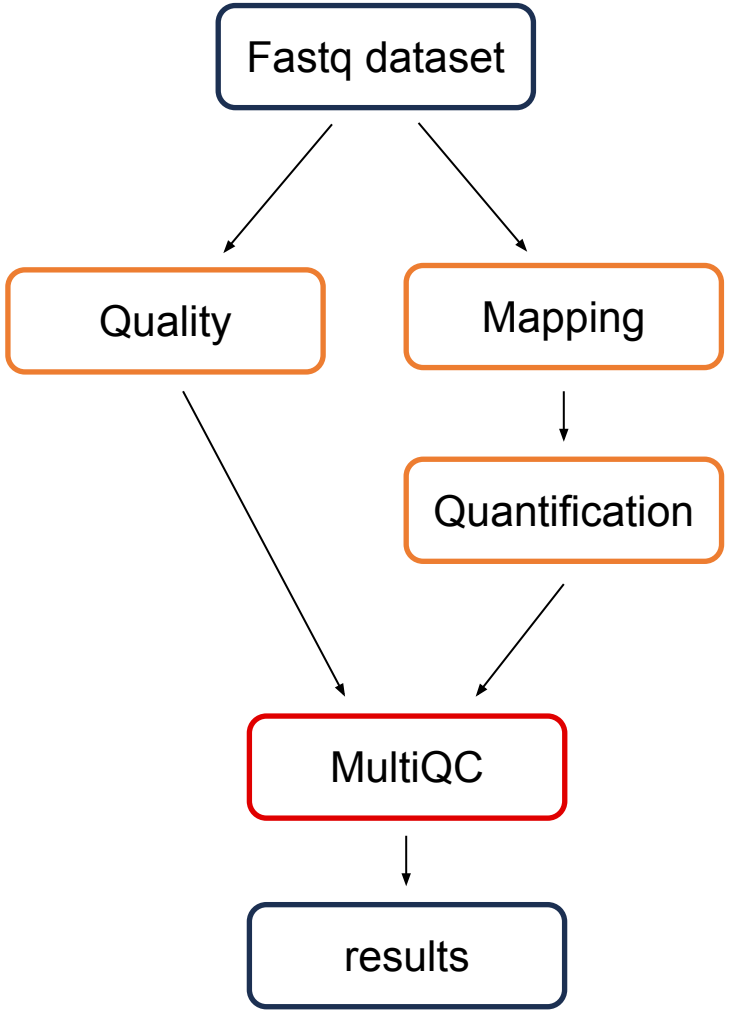
https://snakemake.github.io

Why snakemake ?

- **easy** to use
- is based on **Python** (but know how to code in Python is not required)
- has features for defining the **environment** for each task (running a large number of small third-party tools is current in bioinformatics)
- is easily to be **scaled** from desktop to server, cluster, grid or cloud environments without modification from your single core laptop (ie. develop on laptop using a small subset of data, run the real analysis on a cluster)

Workflow defined by a set of steps = rules



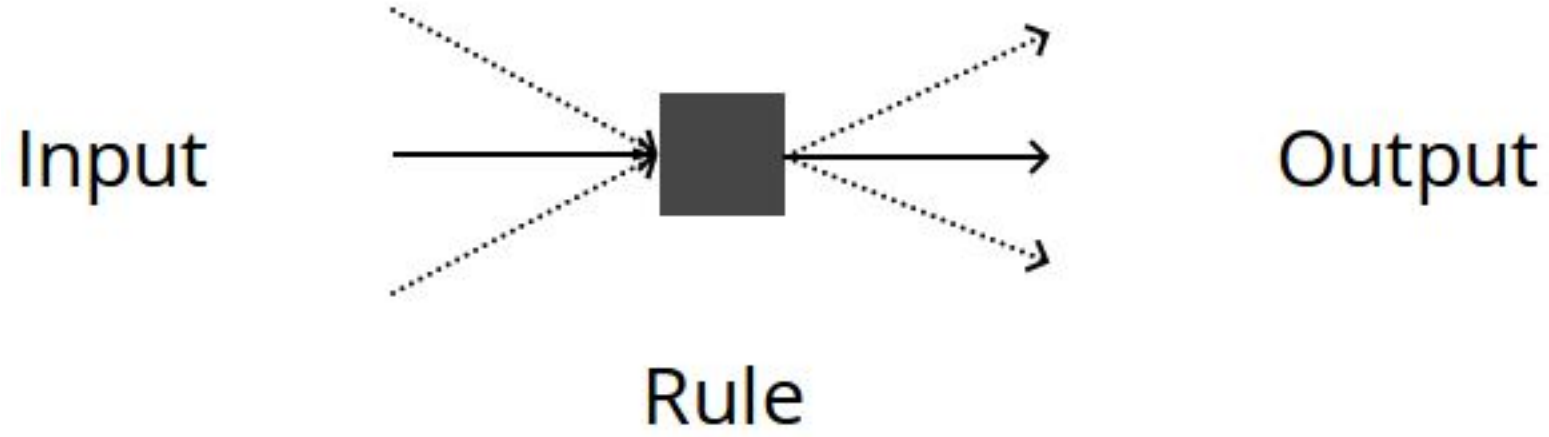


Principles

- a **Snakefile** contains all the rules
- a rule defined a small step (how the files are created)
- snakemake determines the dependencies between rules, it creates a **directed acyclic graph** (DAG)
- Snakemake **only re-runs jobs if one of the input files is newer than one of the output files or one of the input files will be updated by another job**.
- You can fix the computing resources needed per rule (threads, mem, …)
- Conda and Apptainer environments

Rule system

- one step, one rule



- a rule is defined by it name and may contain **directives**:
  - `input:` one or more files
  - `output:` one or more files
  - Command (NB : `run:` for python / `shell:` for shell, R, bioinformatics tool)
  - Options: parameters, log file, message, ….

Rule example

Rule name

grep.smk

```
1   rule grep_chr:
2       input:
3           "Data/0.tauri_genome.fna"
4       output:
5           "chr_list.txt"
6       shell:
7           "grep '>' {input} > {output}"
8
```

Mots clés

```
$ module load snakemake/8.9.0
```

```
[vcognat@core-login2 FAIR-bioinfo2024]$ snakemake -j1 -s grep.smk
Assuming unrestricted shared filesystem usage.
Building DAG of jobs...
Using shell: /usr/bin/bash
Provided cores: 1 (use --cores to define parallelism)
Rules claiming more threads will be scaled down.
Job stats:
job         count
--------  -------
grep_chr        1
total           1

Select jobs to execute...
Execute 1 jobs...

[Wed Mar 20 15:24:16 2024]
localrule grep_chr:
    input: Data/0.tauri_genome.fna
    output: chr_list.txt
    jobid: 0
    reason: Missing output files: chr_list.txt
    resources: tmpdir=/tmp

[Wed Mar 20 15:24:16 2024]
Finished job 0.
1 of 1 steps (100%) done
Complete log: .snakemake/log/2024-03-20T152416.537653.snakemake.log
```

Rule for mapping

- Write the rule to execute this command line :

```
hisat2  -x hisat2_indexes/Otauri -q -U Data/SRR3099585_chr18.fastq.gz
 -S hisat2/SRR3099585_chr18.sam
```

Rule for mapping

- Write the rule to execute this command line :

```
hisat2  -x hisat2_indexes/Otauri -q -U Data/SRR3099585_chr18.fastq.gz
 -S hisat2/SRR3099585_chr18.sam
```

mapping.smk

```
rule hisat2:
    input:
        "Data/SRR3099585_chr18.fastq.gz"
    output:
        "hisat2/SRR3099585_chr18.sam"
    params:
        "hisat2_indexes/Otauri"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {params} -q -U {input} -S {output}"

```

```
module load hisat2 snakemake
snakemake -j 1 -s mapping.smk
```

Rule with multiple input or output files

- By index

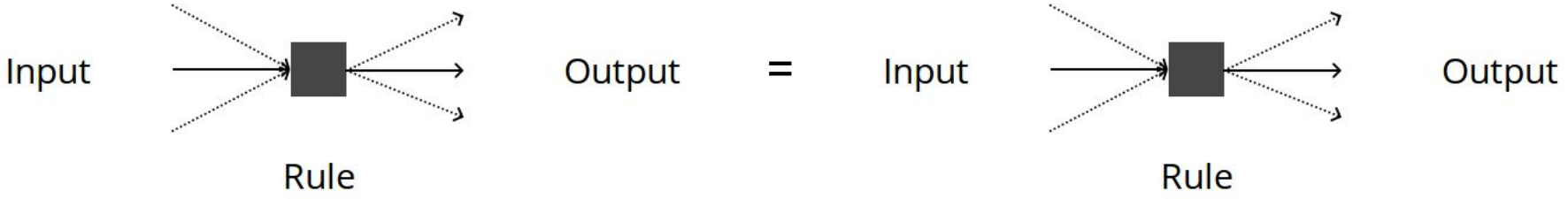
```
rule alignPE:
  input:
    "file_R1.fastq",
    "file_R2.fastq"
  output:
    "alignment.sam"
  shell:
    "hisat2 -1 {input[0]} -2 {input[1]} -x index -S {output}"
```

- By name

```
rule alignPE:
  input:
    R1="file_R1.fastq",
    R2="file_R2.fastq"
  output:
    sam="alignment.sam"
  shell:
    "hisat2 -1 {input.R1} -2 {input.R2} -x index -S {output.sam}"
```

Data flow linkage

- A snakemake workflow links rules thank to the filenames of the rule input and output directives:



- Snakemake rules **order**: the first rule is the default target rule and specifies the result files
- Snakemake creates a **DAG** (directed acyclic graph) corresponding to the rules linkage

The Snakefile

- The **Snakefile** is the **text file** that encodes the rules, and so the workflow. The command snakemake runs the workflow encoded in the Snakefile file.
- You can get a snakefile:
  - from github, your colleagues, ...
  - snakemake "core" (nf-core equivalent) : https://snakemake.github.io/snakemake-workflow-catalog/ (2k pipelines, 236 testés)
  - compose with snakemake wrappers
  - by using a Nextflow workflow! (integration via snakemake-wrappers)
  - create from scratch
- To run the workflow for one input: snakemake –s Snakefile

Snakemake run options

- `-s mySnakefile` to change the default snakefile name
- dry-run, do not execute anything, display what would be done: `-n --dryrun`
- print the shell command: `-p --printshellcmds`
- print the reason for each rule execution: `-r --reason`
- print a summary and status of rule: `-D`
- Limit the number of jobs in parallel: `-j 1` (cores: `-c 1`) [⚠️ mandatory]
- Go on with independant jobs if a job fail : `-k --keep-going`

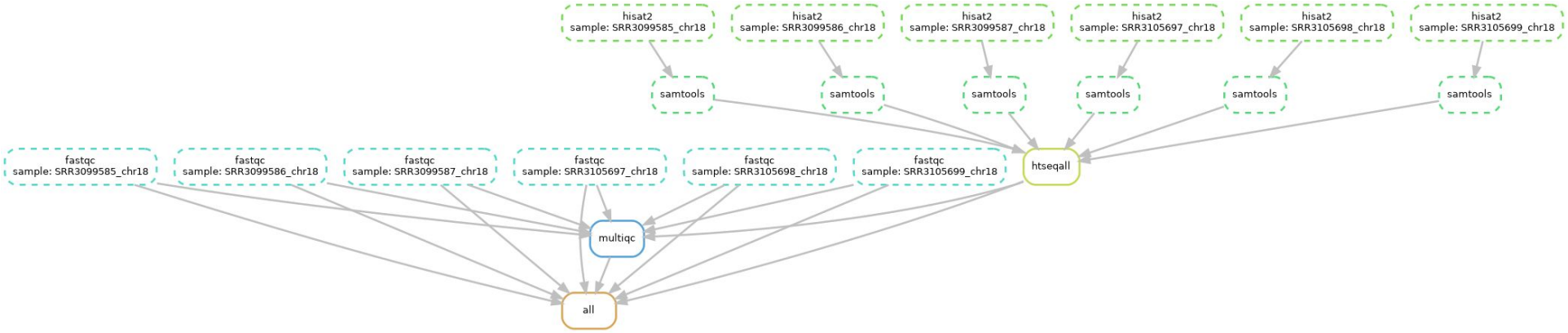
➢ all snakemake options

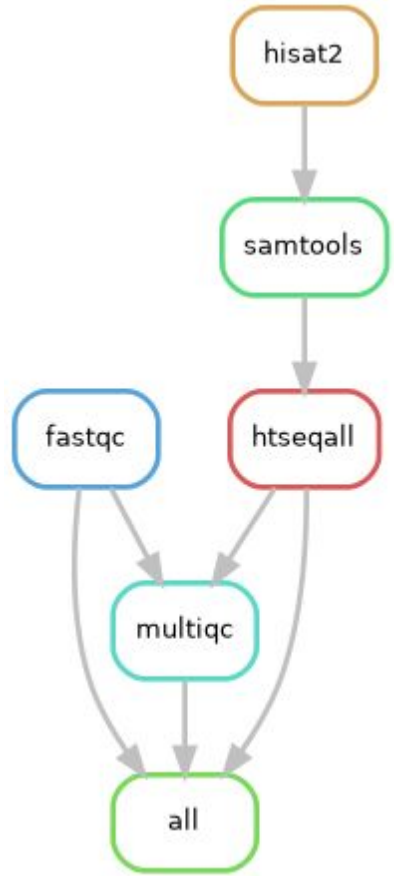
Snakemake output options

- to automatically create **HTML reports** (`--report report.html`) with runtime statistics, a visualization of the workflow topology, used software and data provenance information
- use the `--archive` option (**need git**) to save your project
- complete workflow (`--dag`) or rules dependencies (`--rulegraph`) **visualizations** (with the dot tool of the graphviz package)

Snakemake output options

```
snakemake --dag -s mySmk | dot -Tpng > mySmk_dag.png
snakemake --rulegraph -s mySmk | dot -Tpng > mySmk_rule.png
```





Snakefile structure

- Include the python libraries to use (ex: pandas, snakemake.utils, snakemake.logging)
- Contains all the global variables
- The rules
- 1st = default rule rule = 1st execute
  - Name: `rule all:`
  - Contains all the build-targets (similar to GNU make)

```
rule all:
  input: "SRR3099585_fastqc.hml"," SRR3099585_fastqc.zip"
```

Snakefile: input/output specifications

- enumerated:

```
rule all:
  input: "SRR3099585_fastqc.hml","SRR3099586_fastqc.html"
```

- python list & wildcards:

```
DATASETS=["SRR3099585","SRR3099586"]
rule all:
  input: ["{dataset}_fastqc.hml".format(dataset=dataset) for dataset in
DATASETS]
```

- expand() & wildcards:

```
DATASETS=["SRR3099585","SRR3099586"]
rule all:
  input: expand("{dataset}_fastqc.hml",dataset=DATASETS)
```

Snakefile: generalization with wildcards

- Snakemake use *wildcards* allow to replace parts of filename:
  - reduce hardcoding: more flexible input and output directives, work on new data without modification
  - are automatically resolved (ie. replaced by regular expression ".+" in filenames)
  - are writing into {}
  - are specific to a rule
- A same file can be accessed by different matchings:
  Ex. with the file 101/file.A.txt :

```
rule one : output : "{set}1/file.{grp}.txt" # set=10, grp=A
rule two : output : "{set}/file.A.{ext}" # set=101, ext=txt
```

(more on wildcards in the snakemake documentation)

Wilcards example

- without wildcards, uniprot_wow.smk:

```
rule get_prot:
  output: "P10415.fasta", "P01308.fasta"
  run :
    shell("wget https://www.uniprot.org/uniprot/P10415.fasta")
    shell("wget https://www.uniprot.org/uniprot/P01308.fasta")
```

- with wildcards, uniprot_wiw.smk:

```
rule all:
  input: "P10415.fasta", "P01308.fasta"

rule get_prot:
  output: "{prot}.fasta"
  shell: "wget https://www.uniprot.org/uniprot/{wildcards.prot}.fasta"
```

Global wildcards : get input file names from the file system

To deduce the identifiers (eg. IDs) of files in a directory, use the inbuilt **`glob_wildcards`** function, eg.:

```
IDs, = glob_wildcards("dirpath/{id}.fastq.gz")
```

`glob_wildcards()` matches the given pattern against the files present in the system and thereby infers the values for all wildcards in the pattern (`{id}` here).

📌 **Hint:** Don't forget the **coma** after the name (left hand side, IDs here).

Exercice

Rewrite mapping.smk to use list and/or glob_wildcards
to map all the samples on the genome

Example with python list

mapping-list.smk

```
1   SAMPLES = ["SRR3099585_chr18","SRR3099586_chr18","SRR3099587_chr18"]
2   HISAT2idx = "hisat2_indexes/Otauri"
3
4   rule all:
5     input:
6       expand("results/mapping/{sample}.sam", sample=SAMPLES)
7
8   rule hisat2:
9     input:
10      "Data/{sample}.fastq.gz"
11    output:
12      "results/mapping/{sample}.sam"
13    message: """--- Mapping with hisat2 {input}."""
14    shell:
15      "hisat2 -x {HISAT2idx} -q -U {input} -S {output}"
16
```

Example with global wildcards

mapping-globwild.smk

```
1   SAMPLES, = glob_wildcards("Data/{sample}.fastq.gz")
2   HISAT2idx = "hisat2_indexes/Otauri"
3
4   rule all:
5     input:
6       expand("results/mapping/{sample}.sam", sample=SAMPLES)
7
8   rule hisat2:
9     input:
10      "Data/{sample}.fastq.gz"
11    output:
12      "results/mapping/{sample}.sam"
13    log:
14      "results/mapping/{sample}.log"
15    message: """--- Mapping with hisat2 {input}."""
16    shell:
17      "hisat2 -x {HISAT2idx} -q -U {input} -S {output} 2> {log}"
18
```

Snakemake environment options

Snakemake supports environments on a per-rule basis (created & activated on the fly): **`--software-deployment-method`** OR **`--sdm {apptainer,conda,env-modules}`**

- **conda:**
  - add a `conda:` in the rule definition (*eg.* `conda: myCondaEnv.yml`)
  - run snakemake with the `--sdm conda` OR `--use-conda` option
- **docker:**
  - add a `container:` in the rule definition (*eg.* `container: "docker://biocontainers/fastqc"`)
  - run snakemake with the `--sdm apptainer` OR `--use-apptainer` and `--singularity-args "-B /path/outside/container/:/path/inside/container/"` options
- **module:**
  - add a `envmodules:` in the rule definition (*eg.* `envmodules: "fastqc/0.11.9"`)
  - run Snakemake with the `--sdm env-modules` OR `--use-envmodules` option

Example – Env modules

-envmod.smk ×

```
SAMPLES, = glob_wildcards("Data/{sample}.fastq.gz")
HISAT2idx = "hisat2_indexes/Otauri"

rule all:
    input:
        expand("results/mapping/{sample}.sam", sample=SAMPLES)

rule hisat2:
    input:
        "Data/{sample}.fastq.gz"
    output:
        "results/mapping/{sample}.sam"
    envmodules:
        "hisat2/2.2.1"
    log:
        "results/mapping/{sample}.log"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {HISAT2idx} -q -U {input} -S {output} 2> {log}"
```

```
module purge; module load snakemake
snakemake -j 1  -s 4-envmod.smk --sdm env-modules
```

Example – Conda env

envs.smk

```
rule all:
    input:
        expand("results/mapping/{sample}.sam", sample=SAMPLES)

rule hisat2:
    input:
        "Data/{sample}.fastq.gz"
    output:
        "results/mapping/{sample}.sam"
    envmodules:
        "hisat2/2.2.1"
    conda:
        "envs/hisat2.yml"
    log:
        "results/mapping/{sample}.log"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {HISAT2idx} -q -U {input} -S {output} 2> {log}"
```

hisat2.yml

```
1  name: hisat2
2  channels:
3    - bioconda
4    - conda-forge
5    - defaults
6    - r
7  dependencies:
8    - bioconda::hisat2=2.2.1
```

```
module purge; module load snakemake conda
snakemake -j 1  -s 4-envs.smk --sdm conda
```

Example – use apptainer

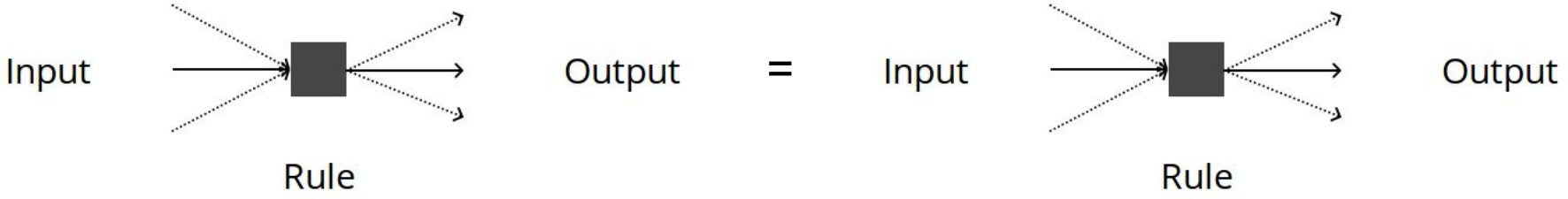
envs.smk

```
SAMPLES, = glob_wildcards("Data/{sample}.fastq.gz")
HISAT2idx = "hisat2_indexes/Otauri"

rule all:
    input:
        expand("results/mapping/{sample}.sam", sample=SAMPLES)

rule hisat2:
    input:
        "Data/{sample}.fastq.gz"
    output:
        "results/mapping/{sample}.sam"
    envmodules:
        "hisat2/2.2.1"
    conda:
        "envs/hisat2.yml"
    container:
        "docker://quay.io/biocontainers/hisat2:2.2.1--hdbdd923_6"
    log:
        "results/mapping/{sample}.log"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {HISAT2idx} -q -U {input} -S {output} 2> {log}"
```

```
module purge; module load snakemake singularity
snakemake -j 1  -s 4-envs.smk --sdm apptainer
```

How to chain the rules

- A snakemake workflow links rules thank to the filenames of the rule input and output directives:




```
rule hisat2:
    input:
        "Data/{sample}.fastq.gz"
    output:
        "results/mapping/{sample}.sam"
    log:
        "results/mapping/{sample}.log"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {HISAT2idx} -q -U {input} -S {outpu
```

```
rule samtools:
    input:
        "results/mapping/{sample}.sam"
    output:
        bam="results/mapping/{sample}.bam",
        bai="results/mapping/{sample}.bam.bai"
    message: """--- Samtools sort and index {input}"""
    shell:
        """
        samtools view -b {input} | samtools sort -o {output.bam} -
        samtools index {output.bam}
        """
```

How to chain the rules

- Since snakemake 7.0
- `rule.rulename.output`

```
rule hisat2:
    input:
        "Data/{sample}.fastq.gz"
    output:
        "results/mapping/{sample}.sam"
    log:
        "results/mapping/{sample}.log"
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -x {HISAT2idx} -q -U {input} -S {output} 2
```

```
rule samtools:
    input:
        rules.hisat2.output
    output:
        bam="results/mapping/{sample}.bam",
        bai="results/mapping/{sample}.bam.bai"
    message: """--- Samtools sort and index {input}"""
    shell:
        """
        samtools view -b {input} | samtools sort -o {output.bam} -
        samtools index {output.bam}
        """
```

`rule.samtools.output.bam`

Exercice

Write the samtools rule and execute snakemake

Snakemake wrappers

- https://snakemake-wrappers.readthedocs.io/

- A collection of reusable wrappers that allow to quickly use popular command line tools from snakemake rules and workflows.
- Based on conda environment: `snakemake --sdm conda`
- Recommended by snakemake team

Exercice

Write the fastqc rule with the snakemake wrapper

```
module purge; module load snakemake conda
snakemake -j 1  -s fastqc.smk --sdm conda
```

Exercice – fastqc wrapper

fastqc.smk

```
SAMPLES, = glob_wildcards("Data/{sample}.fastq.gz")

rule all:
    input:
        expand("results/quality/{sample}_fastqc.html", sample=SAMPLES),
        expand("results/quality/{sample}_fastqc.zip", sample=SAMPLES)

rule fastqc:
    input:
        "Data/{sample}.fastq.gz"
    output:
        html="results/quality/{sample}_fastqc.html",
        zip="results/quality/{sample}_fastqc.zip"
    params:
        extra = "--quiet"
    log:
        "logs/fastqc/{sample}.log"
    threads: 1
    resources:
        mem_mb = 1024
    wrapper:
        "v3.7.0/bio/fastqc"
```

Best practices

IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea INVESTIR L'AVENIR

Use a config file & samplesheet file

- **Config file**
  - Contains the hard coding values for the workflow
  - files path
  - Parameters
  - Json or yaml (config.yml)
  - Import in Snakefile with `configfile: "config.yaml"`
  - OR `snakemake --configfile path/to/config.yaml`
- **Sample sheets**
  - tsv format (pandas library)
  - metadata and exp information

Snakemake workflow template

- Standardized structure
- [snakemake-workflow-template]

📌 Each rule will be imported in Snakefile with:

```
include: "rules/module1.smk"
```

```
├── .gitignore
├── README.md
├── LICENSE.md
├── 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
├── config
│   ├── config.yaml
│   └── some-sheet.tsv
├── results
└── resources
```

Resources allocation

- In command line :
  - --resources, --default-resources, --set-threads, --set-resources, --set-default-resources, --directory, …
- In rule
  - threads:
  - ressources:
    - mem_mb ;
    - disk_mb ;
    - runtime ;
    - tmpdir

```
rule hisat2:
    input: …
    output: …
    params: …
    envmodules: …
    conda: …
    container: …
    threads: 2
    resources:
        mem_mb = 2048
    log: …
    message: """--- Mapping with hisat2 {input}."""
    shell:
        "hisat2 -p {threads} -x {params.idx} -q -U {input.fq} -S {output} -
```

Resources allocation

- In workflow file `--workflow-profile WORKFLOW_PROFILE`
  - easier to modify by others
  - overwrite the rule threads and resources
  - workflow/profiles/default/config.yaml

```
resources:
    mem_gb: 16
    threads: 8

set-threads:
    rulename: 3
set-resources:
    rulename:
        mem_mb: 1000

default-resources:
    mem_mb: 2000
    threads: 1
```

global resources

rule resources

default resources

❗❗ Could contain all your favorite parameters:

```
restart-times: "3"
latency-wait: "60"
use-conda: "True"
use-singularity: "False"
jobs: "8"
keep-going: True
```

Snakemake best practices

- Global workflow dependencies (pandas, snakemake plugins)
  - `conda: "envs/global.yaml »`

```
channels:
  - conda-forge
  - bioconda
  - nodefaults
dependencies:
  - pandas=1.0.3
  - snakemake-storage-plugin-s3
```

- Keep python code separate from rules (ex: commons.smk)
  - `rules/commons.smk`
  - Avoid lambda expression inside rules
- Use Snakemake wrappers whenever possible

Snakemake best practices

- Fix minimale version of snakemake

```
from snakemake.utils import min_version
min_version("8.9.0")
```

- Write in log instead of print

```
from snakemake.logging import logger
logger.info('Samples to analyze: ' + " ".join(SAMPLES))
```

- Code quality checker:
  - `snakemake --lint`

run snakemake with slurm executor

- https://snakemake.github.io/snakemake-plugin-catalog/
- in command line

```
snakemake --executor slurm --default-resources slurm_account=<your SLURM account>
slurm_partition=<your SLURM partition>
```

- in profile file workflow/profiles/default/config.yaml

```
executor: "slurm"
default-resources:
  slurm_account: "account_name"
  slurm_partition: "fast"
  runtime: 120  # in minutes
  cpus_per_task: 1
  nodes: 1
  tasks: 1
  mem_mb: 1000
```

Exercice

Complete the workflow with htseq-count and multiqc

Create the config file

Use the snakemake workflow template

Questions ?

IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea INVESTIR L'AVENIR

Links

- https://snakemake.readthedocs.io/
- https://slides.com/johanneskoester/snakemake-tutorial