

FAIR Bioinfo 2023

Introduction to workflow language with Snakemake

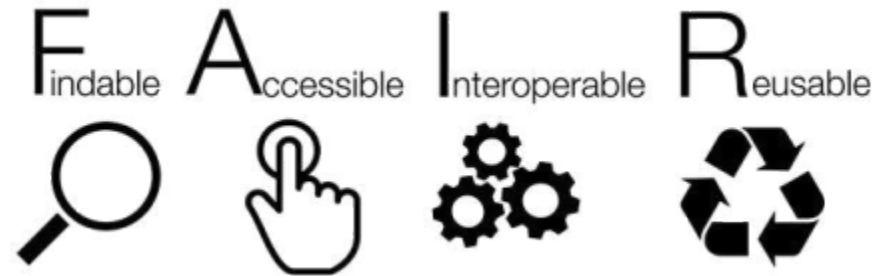
P. Marin, M. Hiriart, P. Ruiz, Nadia Goué
aubi@uca.fr

Université Clermont Auvergne, AuBi, Mésocentre

Du 10 au 13 & 14 juillet 2023

- Workflow introduction
- Snakemake introduction
- A little more on snakefile and configuration files

Interest in workflow management



- ❑ The **standardization**, **portability**, and **reproducibility** of analysis pipelines is a renowned problem within the bioinformatics community.
- ❑ It had struggled with issues of **reproducibility** and **data provenance**.
- ❑ Being able to reproduce scientific results is the central tenet of the scientific method.

Moving towards **FAIR** research methods **in data-driven science** is complex

What is a workflow ?

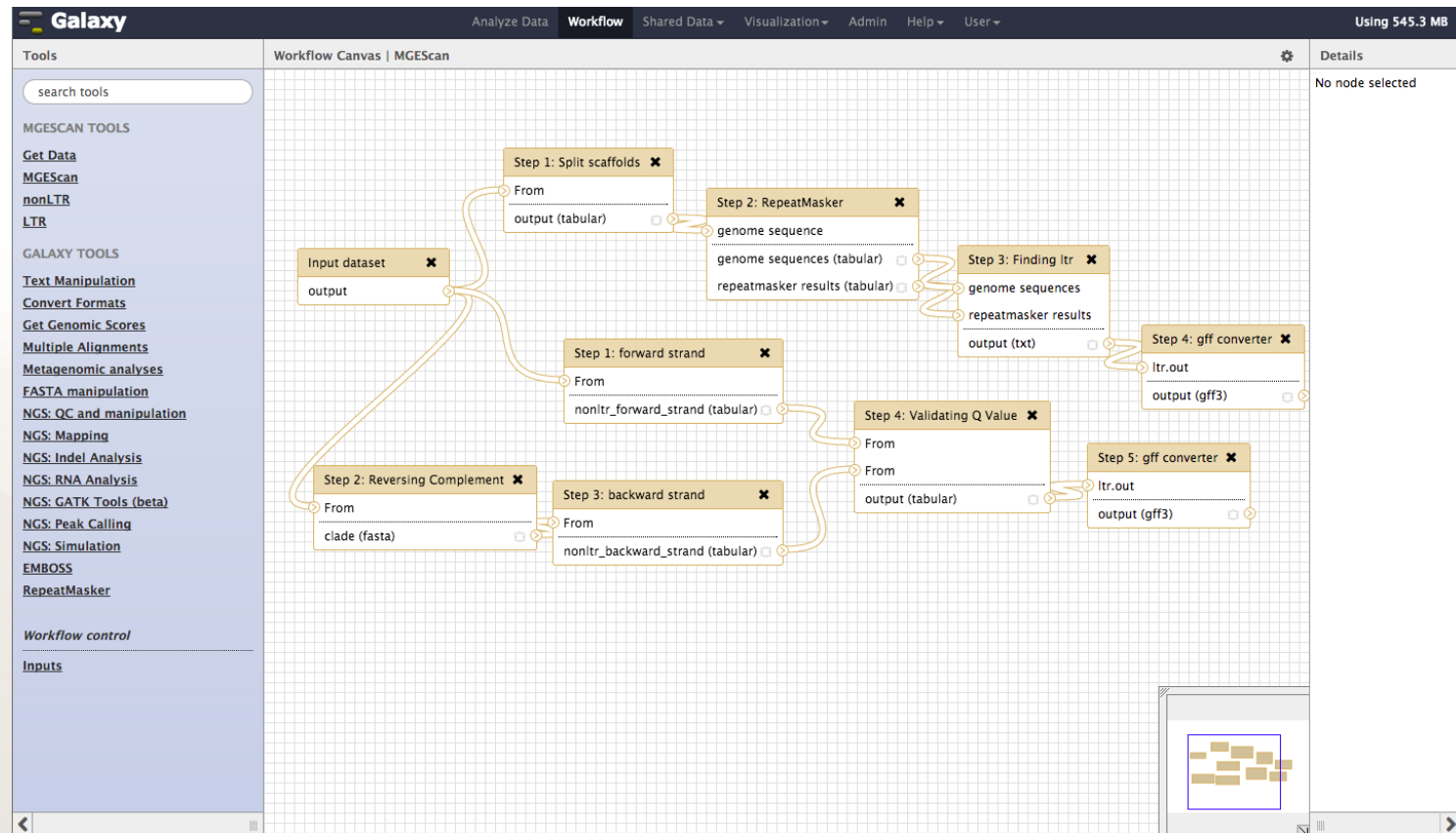
A pool of commands, progressively linked by the **treatments**,
from the **input** data towards the **results**

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

- ✓ with a multi-cores PC or a computational cluster (ex. ~ 3 100 cores)
- ✓ one (or more) core can be attributed to one workflow

What is a workflow ?

- From **informatics** point of view : **Group of programs** executed **in series**
so that the output of 1 program is used as input for the next one.
- From a **bioinformatics** point of view : a workflow is a set of pipelines



What is a workflow ?

- ❑ From **informatics** point of view : **Group of programs** executed **in series**
so that the output of 1 program is used as input for the next one.
- ❑ From a **bioinformatics** point of view : a workflow is a set of pipelines



What are workflow management systems ?

- ❑ Many workflow management systems, many forms

<https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems>

- ❑ **command line**: shell (needs competences into scripting)
- ❑ **rule**: snakemake, c-make, nextflow, ...
- ❑ **graphic interface**: Galaxy, Taverna, Keppler, ...



Reproducibility
Manage parallelization



Learning effort

What is Snakemake ?

Workflow manager

❑ Supports for **dependencies**

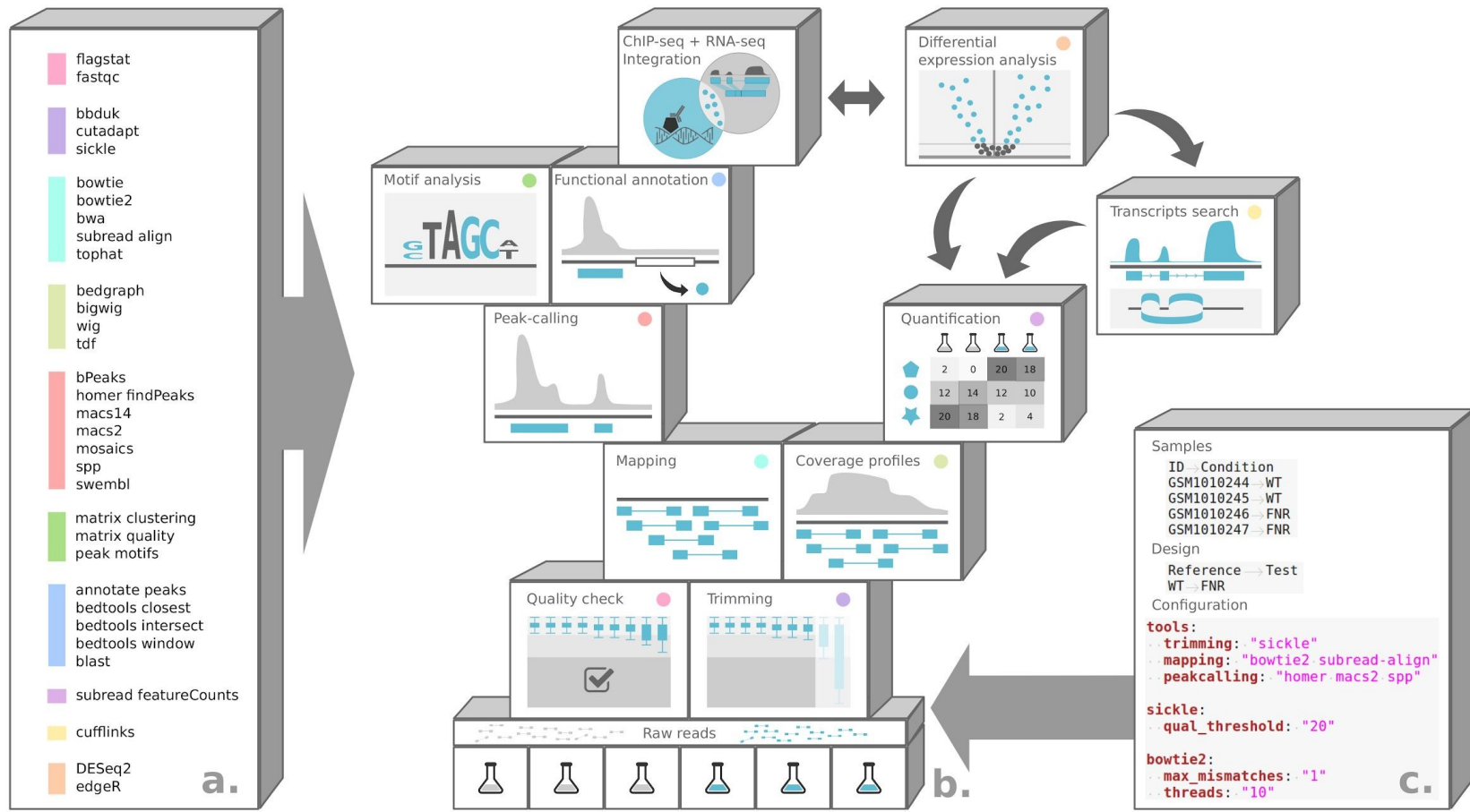
- ✓ Refers to upstream files (or tasks) that downstream transformation steps require as input.
- ✓ When a dependency is updated, associated downstream files should be updated as well.

❑ Support for **reentrancy**

- ✓ Reentrancy is the ability of a program to continue where it left off if interrupted.
- ✓ It's obviating the need to restart from the beginning of a process.

What is Snakemake ?

Reproducible and scalable analysis



Rioualen *et al.* 2017, <https://doi.org/10.1101/165191>, Rioualen *et al.* 2019 <https://doi.org/10.1002/cpbi.72>

The Snakemake rule

- Hybrid between
 - ✓ Python (Snake)
 - ✓ GNU Make, a rule-based automation tool
- Understanding the Rule concept. It has:
 - ✓ understandable name
 - ✓ input, list one or more filenames
 - ✓ output, list one or more filenames
 - ✓ command (run: for python ; shell: for shell, R, ...)
 - ✓ optional directives: params: , message: , log: , threads: , ...

```
rule NAME:
    input:
        "path/to/inputfile",
        "path/to/other/inputfile"
    output:
        "path/to/outputfile",
        "path/to/another/outputfile"
    shell: "some_command {input} {output}"
```

The data flow linkage and rules order

- ❑ **Snakemake** workflow **links** rules thanks to input and output filenames
- ✓ **Until v6.15.0:** the first rule (all, target, ...) need to specify the result files and the next rules describe how to achieve them
- ✓ **From v6.15.0:** any rule can be the default target rule

```
rule all:  
    input: "P10415.fasta"  
  
rule get_prot:  
    output: "P10415.fasta"  
    shell: "wget https://www.uniprot.org/uniprot/P10415.fasta"
```

The data flow linkage and rules order

- **Snakemake** automatically checks that everything is up-to-date:
 - ✓ Create a **directed acyclic graph (DAG)**, linking rules with inputs and outputs
 - ✓ Starts with the **last output** result files of the DAG
 - ✓ Since output files do not exist or have to be re-created, snakemake **goes back** through the workflow
 - ✓ Output files have to be re-created when the input file **timestamp** is newer than the output file one
 - ✓ And from this point, Snakemake goes on through the workflow and apply rules

Generalization with wildcards

- Snakemake use **wildcards** to replace parts of filename:
 - ✓ Reduce hardcoding: more flexible, work with new data without modification
 - ✓ Are automatically resolved
 - ✓ Are written into {}
 - ✓ Are specific to a rule

- Examples

Rule read_firstlines:

input:

"101/file.A.txt"

output:

"{set}1/{file}.{grp}.txt"

shell:

"head -n 10 {input} > {output}"

=> set = 10, grp = A

Rule read_firstlines:

input:

"101/file.A.txt"

output:

"{set}/{file}.A.{ext}"

shell:

"head -n 10 {input} > {output}"

=> set = 101, ext = txt

Input / output specifications

□ Enumerated

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

□ Python list & wildcard

```
DATASETS=["P10415","P01308"]  
  
rule all:  
    input:  
        ["{dataset}.fasta" for dataset in DATASETS]
```

□ expand() & wildcards

```
DATASETS=["P10415","P01308"]  
rule all:  
    input: expand("{dataset}.fasta",dataset=DATASETS)
```

Snakemake Options

□ Using a conda environment

Snakemake supports using explicit conda environments on a per-rule basis:

- ✓ directive in the rule definition:

```
rule NAME:
  input:
  output:
  conda: my_rule_env.yml
  shell:
```

- ✓ Run Snakemake with the `--use-conda` option
- ✓ The specified environment will be created and activated on the fly by Snakemake and the rule will then be run in the conda environment.

Snakemake Options

□ Configuration file

- ✓ contains all hard-coding values of the snakefile (paths to files, core numbers, parameter values, etc)
- ✓ Is a yml or json file and call the defined items with `config["myItem"]`
- ✓ Run with the `--configfile myConfig.yml` Snakemake option
- ✓ Or add the directive `configfile: myConfig.yml` at the beginning of the snakefile

config.yaml

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

snakefile

```
configfile: "config.yaml"  
  
rule NAME:  
  input:  
    expand("{sample}.fastq", sample=config[samples])  
  output:  
  shell:
```


Snakemake Options

□ Some other useful options

- ✓ **dry-run**, no workflow execution, display only what would be done: **-n** **-dryrun**
- ✓ print the shell command: **-p** **-printshellcmds**
- ✓ print a **summary** and **status** of rule: **-D** **-summary**
- ✓ limit the **number of jobs** in parallel: **-j** 1 (cores: **-c** 1)
- ✓ automatically create **HTML reports** (**--report** report.html) containing runtime statistics, a visualization of the workflow topology, used software and data provenance information (need to add the jinja2 package as a dependency)
- ✓ **DAG visualization** with ``dot`` tool (graphviz package), to create diagrams of the complete workflow **--dag** or the rules dependencies **--rulegraph**

References

Amstutz P., et al. 2022. Existing Workflow systems. Common Workflow Language wiki, GitHub.
<https://s.apache.org/existing-workflow-systems> updated 2022-08-30, accessed 2022-08-30.

Leipzig 2017. A review of bioinformatic pipeline frameworks. doi: 10.1093/bib/bbw020

Köster and Rahmann, 2012. Snakemake—a scalable bioinformatics workflow engine.
<https://doi.org/10.1093/bioinformatics/bts480>

Köster and Rahmann, 2018. Snakemake—a scalable bioinformatics workflow engine.
<https://doi.org/10.1093/bioinformatics/bty350>

LeipzigStallma and McGrath 1991. The GNU Make Reference Manual. ISBN-10:168092155X

Rioualen et al. 2017. SnakeChunks : modular blocks to build Snakemake workflows for reproducible NGS analyses. <https://doi.org/10.1101/165191>

Rioualen et al. 2019. Integrating Bacterial ChIP-seq and RNA-seq Data With SnakeChunks.
<https://doi.org/10.1002/cpbi.72>

Workflow PRACTICE

Snakemake for a 2-steps workflow on Biosphere BioPipes VM