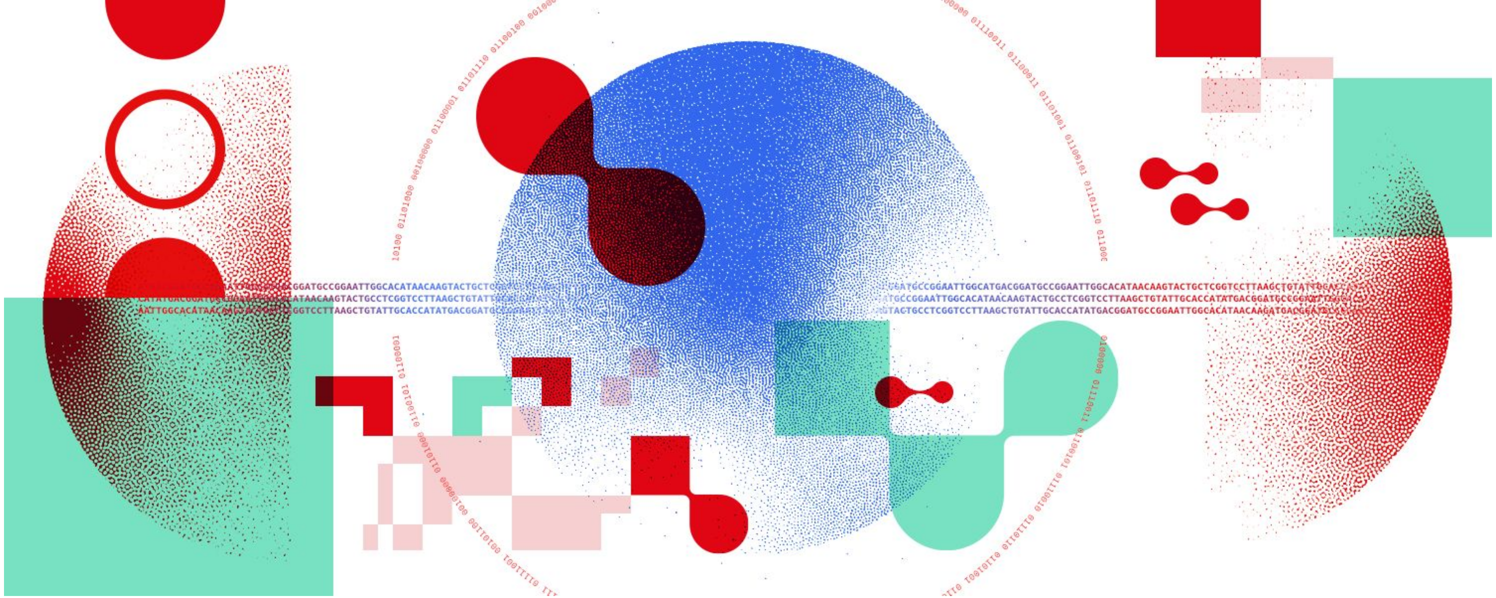


Snakemake for reproducible research

Running Snakemake in an HPC environment

Antonin Thiébaut & Rafael Riudavets Puig
antonin.thiebaut@chuv.ch
Rafael.RiudavetsPuig@empa.ch

Centre hospitalier
universitaire vaudois

Empa
Materials Science and Technology

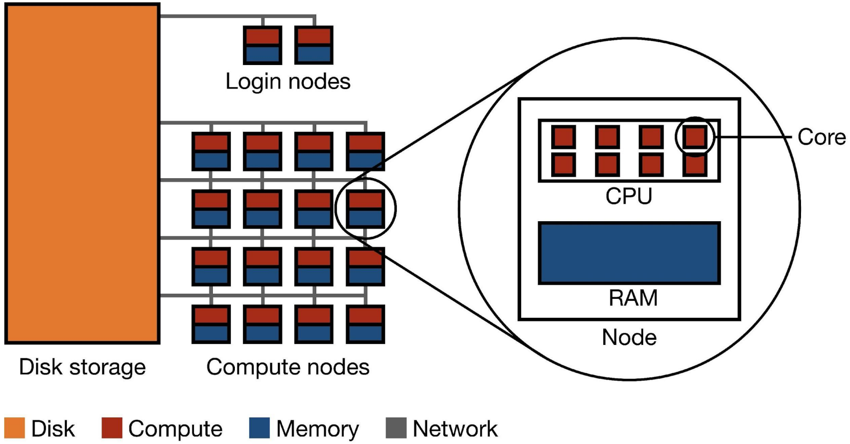
Swiss Institute of
Bioinformatics

- Are you familiar with HPC environments?

- Are you familiar with SLURM?

HPC environments

- Aggregated computing resources to gain performance greater than that of a single workstation, server, or computer
- Used to run computationally heavy processes
- Commonly used simultaneously by multiple users
- Job schedulers (i.e. SLURM) manage jobs sent by all users to ensure a safe and efficient use of the resources
- Extra configuration required:
  - RAM usage, CPUs, run time, …



Statistical Computing and Communication
https://ekatsevi.github.io/statistical-computing/hpc-basics.html

Local versus remote execution

Local

```
python myscript.py
```

Remote

```
sbatch \
    --account=account_name \
    --partition=partition_name \
    --cpus-per-task=4 \
    --time=00:00:30 \
    python myscript.py
```

Local versus remote execution

Local

```
python myscript.py
```

Remote

jobscript.sh

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --account=account_name
#SBATCH --partition=partition_name
#SBATCH --cpus-per-task=4
#SBATCH --time=00:00:30

python myscript.py
```

```
sbatch jobscript.sh
```

Running Snakemake in HPC environments

- Snakemake can interact with multiple schedulers to run on clusters and cloud:
  - **AWS**
  - Azure
  - Flux
  - Google Batch
  - HTCondor
  - Kubernetes
  - **LSF**
  - **Slurm**

- Almost no changes required to the rules
  - Scheduler command can take job information from rule definition
  - One key parameter: maximum number of jobs running in parallel: `-j / --jobs`

- Implemented with:
  - v7 and before: `--cluster "<scheduler_name>"` in the Snakemake command
  - v8+: install plugins then `--executor  "<scheduler_name>"` in the Snakemake command

Towards HPC execution

- Checking rule resource requirements → Benchmark directive
- Resource optimisation → Resources directive

Towards HPC execution

- Checking rule resource requirements → Benchmark directive
- Resource optimisation → Resources directive

Checking rule resource requirements: benchmarks

- 'benchmark' is a directive; its value is a path to a benchmark results file for a rule

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    benchmark:
        'benchmarks/renaming.txt'
    shell:
        'mv {input} {output} '
```

Checking rule resource requirements: benchmarks

- 'benchmark' is a directive; its value is a path to a benchmark results file for a rule
- Snakemake will measure **runtime** and **memory usage** for the rule and save it to the file

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    benchmark:
        'benchmarks/renaming.txt'
    shell:
        'mv {input} {output}'
```

Checking rule resource requirements: benchmarks

- 'benchmark' is a directive; its value is a path to a benchmark results file for a rule
- Snakemake will measure **runtime** and **memory usage** for the rule and save it to the file
- Benchmark files must have the **same wildcards as the output**!
- Best practice: put all benchmarks in same folder

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    benchmark:
        'benchmarks/renaming.txt'
    shell:
        'mv {input} {output} '
```

Towards HPC execution

- Checking rule resource requirements → Benchmark directive
- Resource optimisation → Resources directive

Optimising resource usage: memory and runtime

- 'resources' is a directive; its values set the resources available for a job
    - New kind of directive value: pair of <key>=<value>

Optimising resource usage: memory and runtime

- 'resources' is a directive; its values set the resources available for a job
    - New kind of directive value: pair of <key>=<value>
- mem_<unit>
    - Amount of memory needed by the job
    - <unit>: mb, gb, tb…
- runtime_<unit>
    - Amount of wall clock time a job needs to run
    - <unit>: s, m, h, d…

```
rule example:
    input:
        'data/test.txt'
    output:
        'results/modified_test.txt'
    resources:
        mem_gb = 1,
        runtime_h = 1
    shell:
        'command {input} > {output}'
```

Specifying job resources in Snakemake

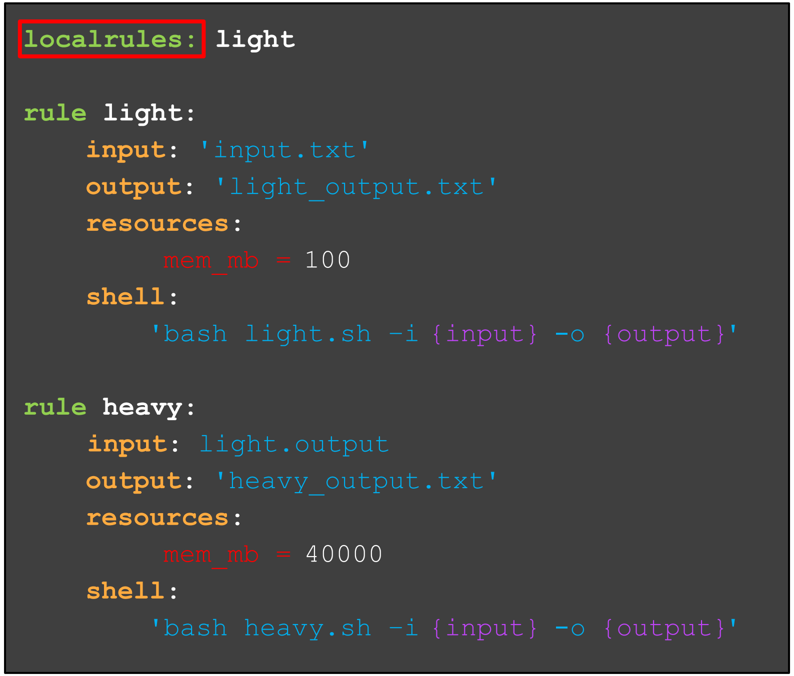
- Job resources are determined as follows:
    - Specifying them by using the resources directive
    - Using default values when no resources specified:
        - RAM: max(2*input.size_mb, 1000)
        - Disk space: max(2*input.size_mb, 1000)
        - Temporary directory: system's tempdir
- Default resources can also be extended when calling Snakemake (i.e. slurm account)

```
rule myrule:
    input:
        'input_{file}.txt'
    output:
        'output_{file}.txt'
    resources:
        mem_mb = 100
    shell:
        'cat {input} > {output}'
```

Rule-specific settings in the Snakefile

- Some jobs are so small that it would be wasteful (and would take longer) to execute on an HPC

- You can define local execution rules using:
    - **localrules** keyword



```
localrules: light

rule light:
    input: 'input.txt'
    output: 'light_output.txt'
    resources:
        mem_mb = 100
    shell:
        'bash light.sh -i {input} -o {output}'

rule heavy:
    input: light.output
    output: 'heavy_output.txt'
    resources:
        mem_mb = 40000
    shell:
        'bash heavy.sh -i {input} -o {output}'
```

Rule-specific settings in the Snakefile

- Some jobs are so small that it would be wasteful (and would take longer) to execute on an HPC

- You can define local execution rules using:
  - **localrules** keyword
  - **localrule** directive

```
rule light:
    input: 'input.txt'
    output: 'light_output.txt'
    resources:
        mem_mb = 100
    localrule: True
    shell:
        'bash light.sh -i {input} -o {output}'

rule heavy:
    input: light.output
    output: 'heavy_output.txt'
    resources:
        mem_mb = 40000
    shell:
        'bash heavy.sh -i {input} -o {output}'
```

Configuration profiles

Configuration profiles

- Preconfigured configuration parameters: resources, executor, sdm…
    - Can manage executor parameters as well:
        - Scripts to submit jobs
        - Scripts to check job status
        - Advanced customisation

- Currently, there are two types of profile:
    - Global: directory stored in ~/.config/snakemake/<profile_name>/
    - Workflow-specific: directory named <profile_name> and containing a config.yaml file.

- The directory contains config files in YAML format.

- Official list of Snakemake profiles here

Configuration profiles

```
./
├─ input_data/
├─ Snakefile
├─ config.yaml
└─ myprofile/
       └─ config.yaml
```

```
executor: cluster-generic
cluster-generic-submit-cmd: 'sbatch –job-name={rule} –cpus-per-task={threads}'
jobs: 10
```

- **executor:** used to indicate how to communicate with the scheduler
    - **cluster-generic** is a Snakemake plugin that handles communication with the scheduler
- **cluster-generic-submit-cmd:** command to use to run the jobs
    - In SLURM, this command is **sbatch** followed by the arguments you want to use
- **jobs:** used to indicate the maximum amount of jobs to run simultaneously
    - Useful to avoid overloading the cluster

Running Snakemake using a profile

- Once set up, running Snakemake using a profile is as simple as:

```
snakemake --profile <path_to_profile_folder>
```

Exercises

- Through the day:
  - Develop a simple RNAseq analysis workflow, from reads (fastq files) to Differentially Expressed Genes (DEG)

- For this session:
  - Benchmark rules to understand resource usage
  - Optimise resource usage
  - Create a configuration profile
  - Run a Snakemake workflow while sending jobs through SLURM

Conclusion

- Snakemake helps with reproducibility:
    - OS, language, software, versions, parameters control via Conda and containers
        - Avoid installation problems!
    - Readability: written in Python, has a well-defined structure
    - Availability: easy to share via WorkflowHub, Snakemake workflow catalog or git
    - **Every command run by Snakemake is saved!**

- And it has many uses:
    - Easily deployable/executable, **locally** or **remotely**
    - Scalable, up to thousands of jobs
        - Easy to parallelise
    - Snakemake can do a lot for you!
    - Beautiful DAG in one command, no more powerpoint or Photoshop!