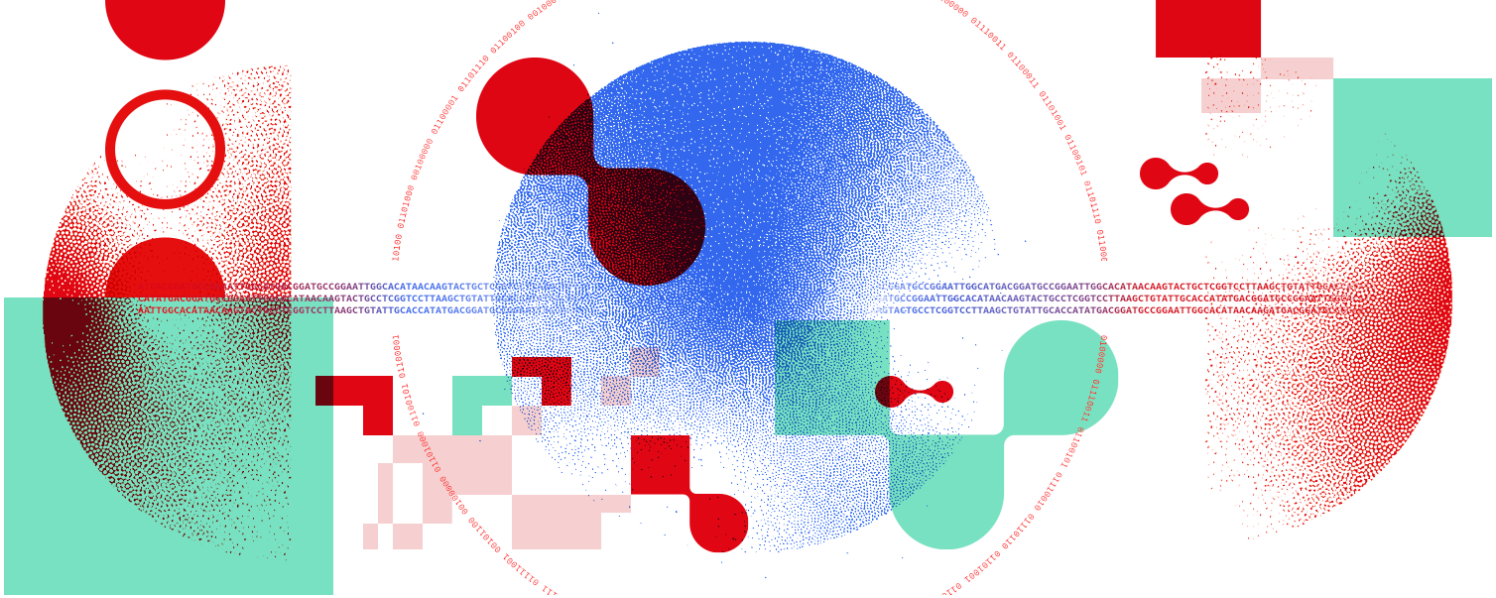

Snakemake for reproducible research

Making Snakemake even more reproducible

Centre hospitalier universitaire vaudois

Antonin Thiébaut

antonin.thiebaut@chuv.ch

SIB Swiss Institute of Bioinformatics

What could we improve? (again)

- Using unknown number of inputs/outputs
- Using scripts from other languages
- Being reproducible

What could we improve? (again)

- Using unknown number of inputs/outputs → Input functions, checkpoints
- Using scripts from other languages → Directives run and script
- Being reproducible → conda/mamba, Docker/Apptainer

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards
  - Number of input files is hard to determine

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards
  - Number of input files is hard to determine
- How to use an input function?
  - Define the function above the rule
  - Use the syntax input: <function_name>
    - No parentheses, no argument

```
def seq_input(wildcards):
    type = wildcards.type
    if type == 'SE':
        return 'data/file1.fq'
    else:
        return ['data/file1.fq', 'data/file2.fq']

rule merge_files:
    input:
        seq_input
    output:
        'results/{type}.txt'
    shell:
        'cat {input} > {output}'
```

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards
  - Number of input files is hard to determine
- How to use an input function?
  - Define the function above the rule
  - Use the syntax input: <function_name>
    - No parentheses, no argument
- Input functions = Python functions
  - Single argument: 'wildcards'
  - Return a file or list of files
  - Can also return a dictionary with input names as keys
    - Use input: unpack(<function_name>) to obtain named inputs

```
def seq_input(wildcards):
    type = wildcards.type
    if type == 'SE':
        return 'data/file1.fq'
    else:
        return ['data/file1.fq', 'data/file2.fq']

rule merge_files:
    input:
        seq_input
    output:
        'results/{type}.txt'
    shell:
        'cat {input} > {output}'
```

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards
  - Number of input files is hard to determine
- How to use an input function?
  - Define the function above the rule
  - Use the syntax input: <function_name>
    - No parentheses, no argument
- Input functions = Python functions
  - Single argument: 'wildcards'
  - Return a file or list of files
  - Can also return a dictionary with input names as keys
    - Use input: unpack(<function_name>) to obtain named inputs
- Functions are evaluated before workflow execution ➔ can't list output files
  - No output functions!

```
def seq_input(wildcards):
    type = wildcards.type
    if type == 'SE':
        return 'data/file1.fq'
    else:
        return ['data/file1.fq', 'data/file2.fq']

rule merge_files:
    input:
        seq_input
    output:
        'results/{type}.txt'
    shell:
        'cat {input} > {output}'
```

```
snakemake --cores 1 results/not_SE.txt
```

⇩

```
{type} = "NotSE"
```

⇩

```
input:
    ['data/file1.fq', 'data/file2.fq']
```

Working after an unknown number of inputs/outputs

- *aka* 'Data-dependent conditional execution' *aka* **checkpoint** (instead of **rule**)
- When:
  - An unknown number of files is generated by a rule
  - Output files are unknown before execution
- Conditional reevaluation of the DAG of jobs based on the outputs content
  - Since DAG is re-evaluated midway ➔ you can't see the whole workflow at the start
- **Very complicated!**

Executing external code in Snakemake

- There are 2 ways to execute external code in Snakemake: run and script

Executing external code in Snakemake: run

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

Executing external code in Snakemake: run

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**
- Problems:
  - Inconvenient for long code
  - No **conda**/**container** directive!!!

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**
- Problems:
  - Inconvenient for long code
  - No **conda**/**container** directive!!!

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    script:
        'first_step.py'
```

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**
- Problems:
  - Inconvenient for long code
  - No **conda**/**container** directive!!!

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    script:
        'first_step.py'
```

- Execute **Python/R/R Markdown/Julia/Rust/bash** code from an external script
- Replaces **shell**/**run**
- Access to directive values and variables, like in **shell**
- Value = path to the script relative to the rule's snakefile

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**
- Problems:
  - Inconvenient for long code
  - No **conda**/**container** directive!!!

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    script:
        'first_step.py'
```

- Execute **Python/R/R Markdown/Julia/Rust/bash** code from an external script
- Replaces **shell**/**run**
- Access to directive values and variables, like in **shell**
- Value = path to the script relative to the rule's snakefile
- Advantages:
  - Great for long code
  - Can use **conda**/**singularity** directive!!!

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces **shell**
- Access to directive values and variables, like in **shell**
- Problems:
  - Inconvenient for long code
  - No **conda**/**container** directive!!!

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    script:
        'first_step.py'
```

first_step.py

```
# Retrieve information from Snakemake
input_file = open(snakemake.input[0])
output_file = open(snakemake.output[0], 'w')
n_lines = snakemake.params.lines

# Process file
for i in range(n_lines):
    output_file.write(input_file.readline())
```

Executing external code in Snakemake: script

- There are 2 ways to execute external code in Snakemake: run and script

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    run:
        input_file = open(input[0])
        output_file = open(output[0], 'w')
        for i in range(params.lines):
            output_file.write(input_file.readline())
```

- Execute **Python** code directly from a Snakefile
- Replaces shell
- Access to directive values and variables, like in shell
- Problems:
  - Inconvenient for long code
  - No conda/container directive!!!

```
rule get_header:
    input:
        'data/file.txt'
    output:
        'results/file_header.txt'
    params:
        lines = 5
    script:
        'first_step.py'
```

first_step.R

```
library(readr)

# Retrieve information from Snakemake
input_path <- snakemake@input[[1]]
output_path <- snakemake@output[[1]]
n_lines <- snakemake@params$lines[1]

# Process file
data <- read_delim(input_path, '\t', n_max=n_lines)
```

Being reproducible with Snakemake and Conda

- What is conda?

Being reproducible with Snakemake and Conda

- What is conda?
  - **Conda/mamba**: open-source, cross-platform, language-agnostic package manager and environment management system
  - **Channels**: field-specific repositories of software
    - Conda-forge: general computation
    - Bioconda: bioinformatics

Being reproducible with Snakemake and Conda

- What is conda?
  - **Conda/mamba**: open-source, cross-platform, language-agnostic package manager and environment management system
  - **Channels**: field-specific repositories of software
    - Conda-forge: general computation
    - Bioconda: bioinformatics
  - Environments defined in YAML files

```yaml
name: python_env
channels:
    - conda-forge
    - bioconda
dependencies:
    - python >= 3.12
    - pandas == 2.2.3
```

py.yaml

Being reproducible with Snakemake and Conda

- Using conda in Snakemake:
  - Snakemake provides integrated package management via Conda to define isolated software environments per rule

Being reproducible with Snakemake and Conda

- Using conda in Snakemake:
  - Snakemake provides integrated package management via Conda to define isolated software environments per rule
  - Directive: **conda**
    - Value: path to the environment file relative to the rule's snakefile

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    conda:
        '../envs/py.yaml'
    shell:
        'mv {input} {output}'
```

Being reproducible with Snakemake and Conda

- Using conda in Snakemake:
  - Snakemake provides integrated package management via Conda to define isolated software environments per rule
  - Directive: **conda**
    - Value: path to the environment file relative to the rule's snakefile
  - Execution parameter:
    - v7 and before: `--use-conda`
    - v8+: `--software-deployment-method` or `--sdm` (shorthand version)

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    conda:
        '../envs/py.yaml'
    shell:
        'mv {input} {output}'
```

```
snakemake --cores 1 --use-conda results/renamed_file.txt
```

```
snakemake --cores 1 --sdm conda
    results/renamed_file.txt
```

Being reproducible with Snakemake and Docker

- What is Docker?

Being reproducible with Snakemake and Docker

- ~~What is Docker?~~

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake:
  - Snakemake provides a container integration: it can automatically spawn a container created from a given image

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake:
  - Snakemake provides a container integration: it can automatically spawn a container created from a given image
  - Directive: **container**
    - Value: URL/path to the image location
    - Handles Docker and Apptainer images
    - **Global OR rule-specific**

```
container: 'docker://geertvangeest/deseq2:v1'

rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    container:
        'docker://geertvangeest/deseq2:v1'
    shell:
        'mv {input} {output}'
```

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake:
  - Snakemake provides a container integration: it can automatically spawn a container created from a given image
  - Directive: **container**
    - Value: URL/path to the image location
    - Handles Docker and Apptainer images
    - **Global OR rule-specific**
  - Execution parameter
    - v7 and before: --use-singularity
    - V8+: --sdm apptainer

```
container: 'docker://geertvangeest/deseq2:v1'

rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    container:
        'docker://geertvangeest/deseq2:v1'
    shell:
        'mv {input} {output}'
```

```
snakemake -cores 1 --use-singularity results/renamed_file.txt
snakemake -cores 1 -sdm=apptainer results/renamed_file.txt
```

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake:
  - Snakemake provides a container integration: it can automatically spawn a container created from a given image
  - Directive: **container**
    - Value: URL/path to the image location
    - Handles Docker and Apptainer images
    - **Global OR rule-specific**
  - Execution parameter
    - v7 and before: --use-singularity
    - V8+: --sdm apptainer
  - Can be combined with conda --sdm conda apptainer
    - Pull the image
    - Create the conda env from **within the container**
  - Containerisation of Conda-based workflows

```
container: 'docker://geertvangeest/deseq2:v1'

rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    container:
        'docker://geertvangeest/deseq2:v1'
    shell:
        'mv {input} {output}'
```

```
snakemake -cores 1 --use-singularity results/renamed_file.txt
snakemake -cores 1 -sdm=apptainer results/renamed_file.txt
```

```
snakemake -cores 1 results/renamed_file.txt --sdm conda apptainer
```

```
snakemake --cores 1 --containerize > Dockerfile
```

Snakemake environments

- Question 6

What is the best setting for Snakemake environments?

- Use package and container managers!
- Same as Snakefile and config files: split things reasonably
  - 1 .smk file ≈ 1 'thematic' module ≈ 1 environment
- Always check for version conflicts

Exercises

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

- For this session:
  - Create and use an input function
  - Run R and Python scripts
  - Deploy a conda environment
  - Deploy a Docker/Singularity container

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!