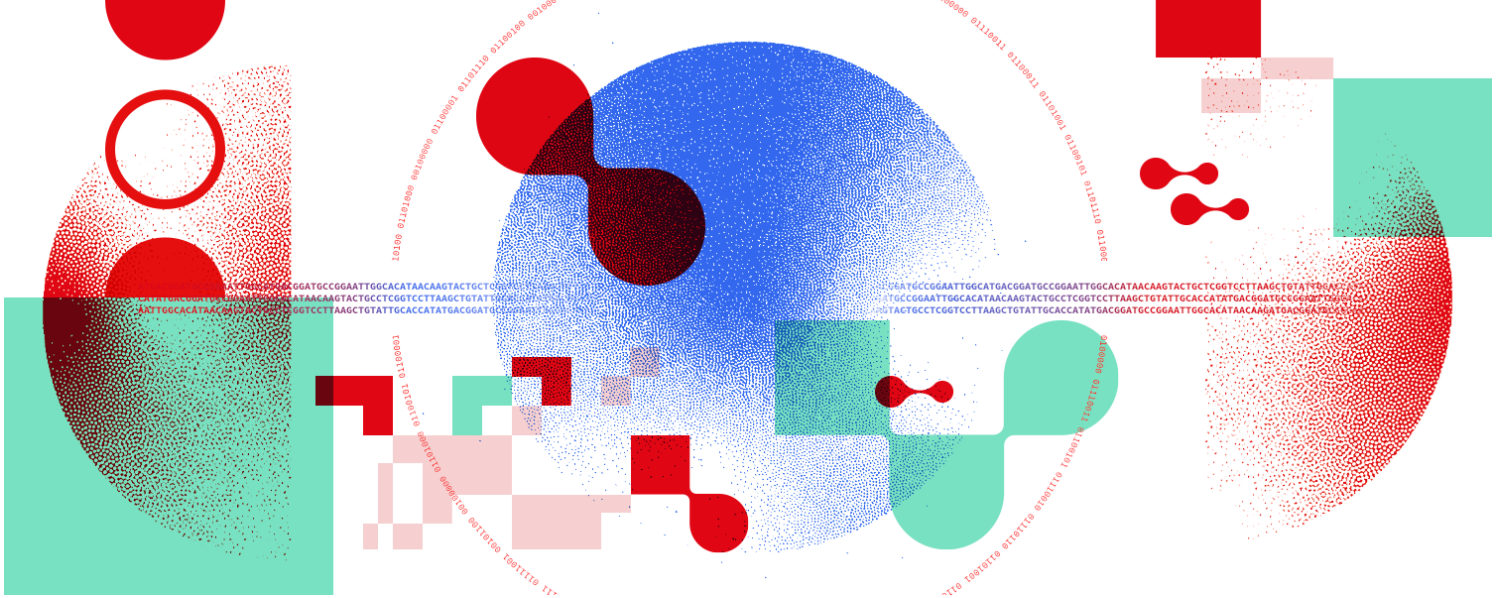

Snakemake for reproducible research

Snakemake, package managers and containers

UNIL | Université de Lausanne

Antonin Thiébaut

antonin.thiebaut@unil.ch

SIB Swiss Institute of Bioinformatics

Config file?

- Question 5

What should appear in a config file?

- Ideally, everything that should not be hard-coded:
  - File locations
  - Sample names and associated information
  - Rule computing resources
  - Etc…

- But it is preferable to use paths to other smaller config files
  - Same as Snakefile and snakefiles
  - Example:
    - Table containing the sample names and information: config/samples_info.tsv
    - In the config file: samples: 'config/samples_info.tsv'
    - Add a function in a Snakefile to parse the table

Several problems…

Several problems… (again)

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

… that can be solved! (again)

- Using scripts from other languages → New directives: run and script
- Using unknown number of inputs/outputs → input/output functions, checkpoint
- Being reproducible → conda/mamba, Docker/Singularity

Executing external code in Snakemake

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

Executing external code in Snakemake

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

Executing external code in Snakemake

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    params:
        lines=5
    script:
        'first_step.py'
```

Executing external code in Snakemake

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    params:
        lines=5
    script:
        'first_step.py'
```

- Execute **Python/R/R Markdown/Julia/Rust/bash** 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

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    params:
        lines=5
    script:
        'first_step.py'
```

- Execute **Python/R/R Markdown/Julia/Rust/bash** 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

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.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**/**singularity** directive!!!

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    params:
        lines=5
    script:
        'first_step.R'
```

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)
```

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards in a non-trivial way
  - The number of input files is not easy to determine manually

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards in a non-trivial way
  - The number of input files is not easy to determine manually
- How to use an input function?
  - Define function above the rule
  - Use syntax input: <function_name>
  - No parentheses, no argument

```
def first_step_input(wildcards):
    sample = wildcards.sample
    if sample == 'sample1':
        return 'data/data1.txt'
    else:
        return 'data/data2.txt'


rule example:
    input:
        first_step_input
    output:
        'results/{sample}.txt'
    shell:
        'cp {input} {output}'
```

```
snakemake --cores 1 results/sample2.txt
```

Working with an unknown number of inputs/outputs

- When:
  - Input files depend on wildcards in a non-trivial way
  - The number of input files is not easy to determine manually
- How to use an input function?
  - Define function above the rule
  - Use 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 executing the workflow ⇒ can’t list output files!

```
def first_step_input(wildcards):
    sample = wildcards.sample
    if sample == 'sample1':
        return 'data/data1.txt'
    else:
        return 'data/data2.txt'


rule example:
    input:
        first_step_input
    output:
        'results/{sample}.txt'
    shell:
        'cp {input} {output}'
```

```
snakemake --cores 1 results/sample2.txt
```

Working after an unknown number of outputs

- *aka* ‘Data-dependent conditional execution’ *aka* **checkpoint** (instead of **rule**)
- When:
  - An unknown number of files is generated by a rule
  - The output files are unknown before execution
- Conditional reevaluation of the DAG of jobs based on the content outputs
  - Since DAG is re-evaluated, you won’t see the whole pipeline at the beginning of a run
- **Very complicated!**

Being reproducible with Snakemake and Conda

- What is conda?

Being reproducible with Snakemake and Conda

- What is conda?
  - **Conda/mamba**: open-source package and environment manager (Windows, macOS, linux)
  - **Channels**: repositories of software, packaged and maintained
    - Conda-forge: lots of general software, often used
    - Bioconda: specifically for bioinformatics software
  - Great tool to manage software in general

Being reproducible with Snakemake and conda

- What is conda?
  - **Conda/mamba**: open-source package and environment manager (Windows, macOS, linux)
  - **Channels**: repositories of software, packaged and maintained
    - Conda-forge: lots of general software, often used
    - Bioconda: specifically for bioinformatics software
  - Great tool to manage software in general
  - Environments can be defined in YAML files

py.yaml

```
name: python_env
channels:
  - conda-forge
  - bioconda
dependencies:
  - python >= 3.10
  - pandas == 1.4.3
```

Being reproducible with Snakemake and conda

- Using conda in Snakemake
  - Snakemake provides a Conda integration: it automatically deploys a conda environment for a rule

Being reproducible with Snakemake and conda

- Using conda in Snakemake
  - Snakemake provides a Conda integration: it automatically deploys a conda environment for a rule
  - Directive **conda**
    - Value = path to the environment file relative to the rule's snakefile
  - Execution parameter --use-conda

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    conda:
        '../envs/py.yaml'
    shell:
        'cp {input} {output}'
```

```
snakemake --cores 1 --use-conda results/first_step.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 Docker integration: it automatically spawns a container created from the given image

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake
  - Snakemake provides a Docker integration: it automatically spawns a container created from the given image
  - Directive **container**
    - Value = URL/path to the image location
    - Handles Docker and Singularity images
  - Execution parameter --use-singularity
  - Can be combined with --use-conda
    - Pull the image
    - Create the conda env from **within the container**

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    container:
        'docker://geertvangeest/deseq2:v1'
    shell:
        'cp {input} {output}'
```

```
snakemake --cores 1 --use-singularity
        results/first_step.txt
```

Being reproducible with Snakemake and Docker

- Using Docker in Snakemake
  - Snakemake provides a Docker integration: it automatically spawns a container created from the given image
  - Directive **container**
    - Value = URL/path to the image location
    - Handles Docker and Singularity images
  - Execution parameter --use-singularity
  - Can be combined with --use-conda
    - Pull the image
    - Create the conda env from **within the container**
  - Create Dockerfile from workflow with conda env

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    container:
        'docker://geertvangeest/deseq2:v1'
    shell:
        'cp {input} {output}'
```

```
snakemake --cores 1 --use-singularity
        results/first_step.txt
```

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

Concluding remarks

- **Reproducibility:**
  - Workflow ⇒ steps clearly defined, commands saved
  - Conda integration ⇒ perfect handling of software installation and versions
  - Self-contained workflow archive ⇒ other people can easily reproduce your analyses (with almost no programming knowledge)
- **Practical use:**
  - Once workflow is build, can be applied to any number of samples
  - Snakemake does a lot for you !
    - Create directory structure
    - Check job completion, restart if needed
    - Fully handles parallelization of jobs
    - Easy handling of logs and benchmarks
  - Portability and scalability: run on the cloud, on HPCs, and on any UNIX machine
  - Beautiful DAG in one command, no more powerpoint !