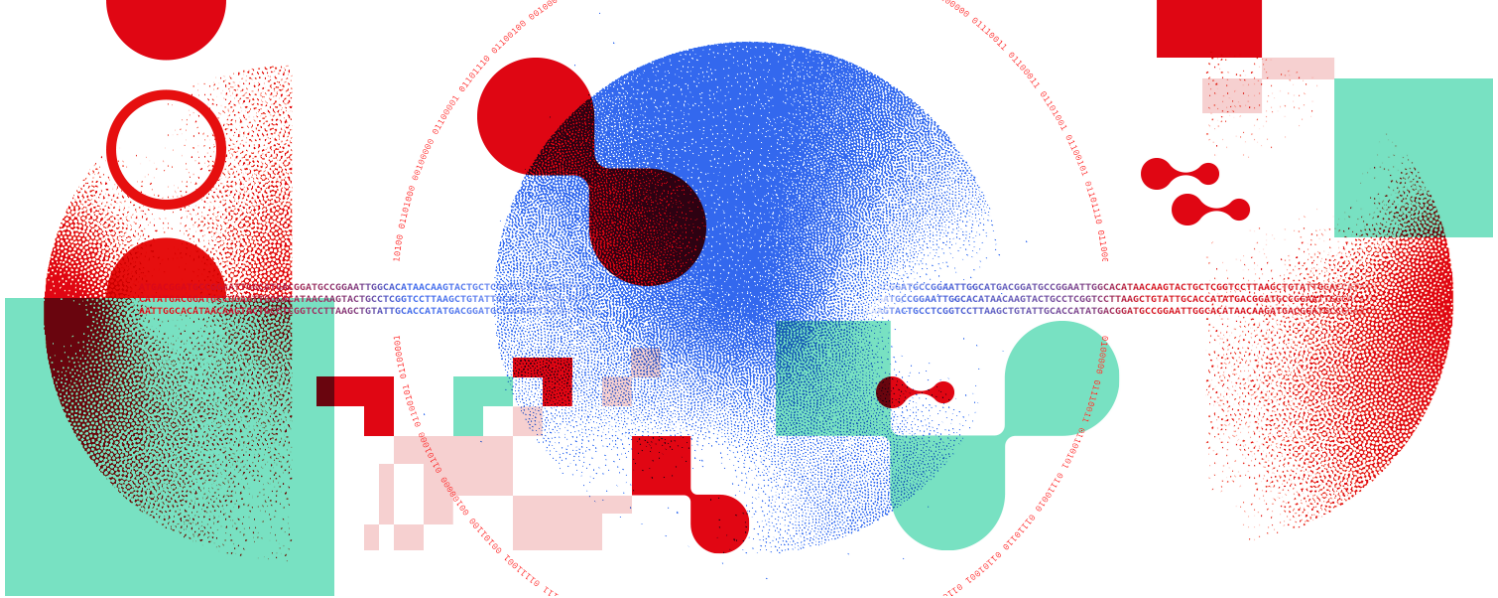


Snakemake for reproducible research

Making a more general-purpose Snakemake workflow

Centre hospitalier universitaire vaudois

Antonin Thiébaut

antonin.thiebaut@chuv.ch

Swiss Institute of Bioinformatics

Pop quiz

```
rule second_step:
    input:
        rules.first_step.output
    output:
        'results/first_step.txt'
    shell:
        'cp {input} {output}'
```

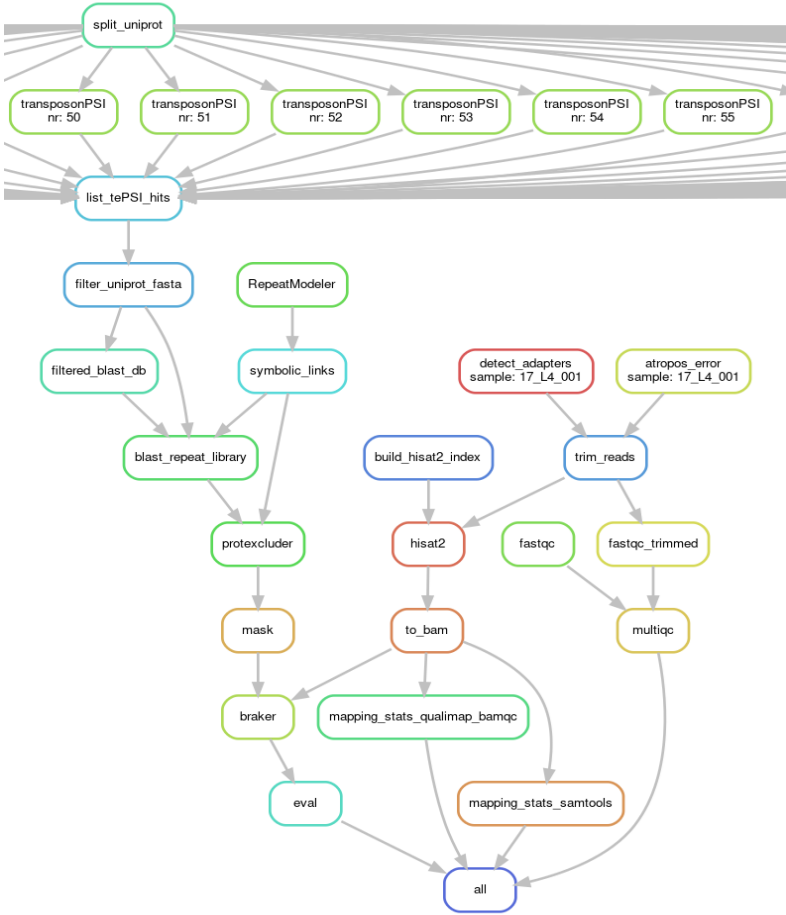
Pop quiz

- Snakemake keyword
- Rule name (user-defined)
- Snakemake directives
- Directives values:
  - Object
  - String (file path)
  - Instruction (command)
  - Numeric values (seen later)
- Mystery syntax?

```
rule second_step:
    input:
        rules.first_step.output
    output:
        'results/first_step.txt'
    shell:
        'cp {input} {output}'
```

Building a Directed Acyclic Graph (DAG)

- Snakemake determines which jobs to run to produce desired outputs



Building a Directed Acyclic Graph (DAG)

- Snakemake determines which jobs to run to produce desired outputs
- Rule can appear more than once, with different wildcards
  - 1 rule + 1 wildcard values = 1 job
- Arrows = dependency between jobs
  - Snakemake runs jobs in any order that doesn't break dependency

Building a Directed Acyclic Graph (DAG)

- Snakemake determines which jobs to run to produce desired outputs
- Rule can appear more than once, with different wildcards
  - 1 rule + 1 wildcard values = 1 job
- Arrows = dependency between jobs
  - Snakemake runs jobs in any order that doesn't break dependency
- DAG = work list, ≠ flowchart
  - No if/else decisions or loops
  - Snakemake runs every job in the DAG exactly once
- DAG does not check shell directives
  - Shell commands are tested during execution (1. Works? 2. Produces expected outputs?)

What is a DAG useful?

- Skip parts of the DAG to avoid recomputing → Save time and resources (CPU, memory, energy, money)
- Change/add inputs to existing analyses without re-running everything
- Resume running a workflow that failed part-way

What could we improve?

What could we improve?

- Using hard-coded file paths
- Having multiple inputs/outputs per rule
- Checking Snakemake behaviour

What could we improve?

- Using hard-coded file paths → Placeholders and wildcards
- Having multiple inputs/outputs per rule → Numbered/named inputs/outputs
- Checking Snakemake behaviour → Log files, benchmarks

Avoiding hard-coded filepaths: placeholders

- Placeholder:
  - A person or thing that occupies the position or place of another person or thing
  - A symbol in a mathematical or logical expression that may be replaced by the name of any element of a set

(From the Merriam-Webster dictionary)

Avoiding hard-coded filepaths: placeholders

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

Avoiding hard-coded filepaths: placeholders

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

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

Avoiding hard-coded filepaths: placeholders

- {input} and {output} are placeholders
- Used in shell directive
- Similar to python f-string
- Snakemake will replace them with appropriate values before running the command
- Many directives can use placeholders: {log}, {benchmark}, {params}…

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

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"

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

“Hard-coded” input and output files

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"

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

“Hard-coded” input and output files

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

“General” input/output files with wildcards

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"

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

"Hard-coded" input and output files

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

"General" input/output files with wildcards

- Enclose wildcard name with curly brackets `{ }`

- How does Snakemake execution work when there are wildcards?

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"

- Wildcards are “resolved” from the target and propagated to other directives
  - Regular expression matching: `.+`
    - “1 or more occurrences of any character except newline”
    - Can be constrained
  - **Using wildcards forces to ask for output(s)**: Snakemake doesn’t guess!
    - Target rules may not contain wildcards

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

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

↓

Snakemake interpretation:

`{sample} = "first_step"`

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"
- Wildcards are “resolved” from the target and propagated to other directives
  - Regular expression matching: `.+`
- Both a workflow and a rule can use multiple wildcards

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

Snakemake interpretation:

```
{sample} = "first"
{treatment} = "step"
```

```
rule example:
    input:
        'data/{sample}_{treatment}.tsv'
    output:
        'results/{sample}_{treatment}.txt'
    shell:
        'echo {wildcards.sample};'
        'cp {input} {output}'
```

Making more general-purpose rules: wildcards

- Wildcards ≈ Snakemake "variables"
- Wildcards are “resolved” from the target and propagated to other directives
  - Regular expression matching: `.+`
- Both a workflow and a rule can use multiple wildcards
- Input and output files do not have to share the same wildcards
- **All outputs/logs… created by a rule must have same wildcards!**

```
rule example:
    input:
        'data/{sample}.tsv'
    output:
        'results/{sample}_{treatment}.txt'
    shell:
        'echo {wildcards.sample};'
        'cp {input} {output}'
```

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

Snakemake interpretation:

input = ‘data/first.tsv’

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs
- Don’t forget the comma!

```
rule example:
    input:
        'data/first_step1.tsv',
        'data/first_step2.tsv'
    output:
        'results/first_step.txt'
    shell:
        'cat {input} > {output}'
```

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs
- Don't forget the comma!

Input directive values are **unpacked** (replaced by a space-separated list)

```
rule example:
    input:
        'data/first_step1.tsv',
        'data/first_step2.tsv'
    output:
        'results/first_step.txt'
    shell:
        'cat {input} > {output}'
```

```
shell:
    cat data/first_step_1.tsv data/first_step_2.tsv > results/first_step.txt
```

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs
- Don't forget the comma/semicolon!
- Inputs can be accessed by their positional index: input[n]
  - **Numbering starts at 0**

```
rule example:
    input:
        'data/first_step1.tsv',
        'data/first_step2.tsv'
    output:
        'results/first_step.txt'
    shell:
        'cat {input[0]} > {output}; '
        'cat {input[1]} >> {output}'
```

Commands are concatenated

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs
- Don't forget the comma!
- Inputs can be accessed by their positional index: input[n]
  - **Numbering starts at 0**

```
rule example:
    input:
        'data/first_step1.tsv',
        'data/first_step2.tsv'
    output:
        'results/first_step.txt'
    shell:
        '''
        cat {input[0]} > {output}
        cat {input[1]} >> {output}
        '''
```

Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs
- Don't forget the comma!
- Inputs can be accessed by their positional index: input[n]
  - **Numbering starts at 0**
- Named input can be accessed by their names: input.input_name
  - You cannot mix named and unnamed inputs

```
rule example:
    input:
        input_1='data/first_step1.tsv',
        input_2='data/first_step2.tsv'
    output:
        'results/first_step.txt'
    shell:
        '''
        cat {input.input_1} > {output}
        cat {input.input_2} >> {output}
        '''
```

Creating rules with multiple inputs/outputs

- Outputs work just like inputs
  - Separated by ‘,’
  - Can be named
  - Can be accessed by positional index or by name

- All outputs need to be generated or the job will fail

```
rule example:
    input:
        input_1='data/first_step1.tsv',
        input_2='data/first_step2.tsv'
    output:
        output_1='results/first_step1.txt',
        output_2='results/first_step2.txt'
    shell:
        '''
        cat {input.input_1} > {output.output_1}
        cat {input.input_2} > {output.output_2}
        '''
```

```
snakemake --cores 1 results/first_step_1.txt
```

→ results/first_step_1.txt, results/first_step_2.txt

Checking Snakemake behaviour

- Producing log files
- Benchmarking rules

Checking Snakemake behaviour: log files

- ‘log’ is a directive; its value is a path to a log file for a rule

Checking Snakemake behaviour: log files

- ‘log’ is a directive; its value is a path to a log file for a rule
  - Can be accessed with a placeholder in ‘shell’: {log}

- Logs still need to be handled manually for each command, but Snakemake automatically creates the directory in the log file path

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    log:
        'logs/first_step.log'
    shell:
        'cp {input} {output} 2> {log}'
```

Checking Snakemake behaviour: log files

- ‘log’ is a directive; its value is a path to a log file for a rule
  - Can be accessed with a placeholder in ‘shell’: {log}
- Logs still need to be handled manually for each command, but Snakemake automatically creates the directory in the log file path
- Log files must have the **same wildcards as the output**!
- Best to regroup logs in a ‘logs’ folder

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    log:
        'logs/first_step.log'
    shell:
        'cp {input} {output} 2> {log}'
```

Checking Snakemake behaviour: benchmarks

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

Checking Snakemake behaviour: benchmarks

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

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    benchmark:
        'benchmarks/first_step.txt'
    shell:
        'cp {input} {output}'
```

Checking Snakemake behaviour: benchmarks

- ‘benchmark’ is a directive; its value is a path to a benchmark results file for a rule
- Snakemake will automatically 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 to regroup benchmarks in a ‘benchmarks’ folder

```
rule example:
    input:
        'data/first_step.tsv'
    output:
        'results/first_step.txt'
    benchmark:
        'benchmarks/first_step.txt'
    shell:
        'cp {input} {output}'
```

Exercises

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

- For this session:
  - Use placeholders and wildcards
  - Use multiple inputs and outputs
  - Check workflow behaviour
  - Visualise a DAG