

# Snakemake for reproducible research

Making a more general-purpose Snakemake workflow



Centre hospitalier  
universitaire vaudois

Antonin Thiébaud

[antonin.thiebaut@chuv.ch](mailto:antonin.thiebaut@chuv.ch)



Swiss Institute of  
Bioinformatics

# Pop quiz

```
rule rename_file:  
    input:  
        rules.create_file.output  
    output:  
        'results/renamed_file.txt'  
    shell:  
        'mv {input} {output}'
```

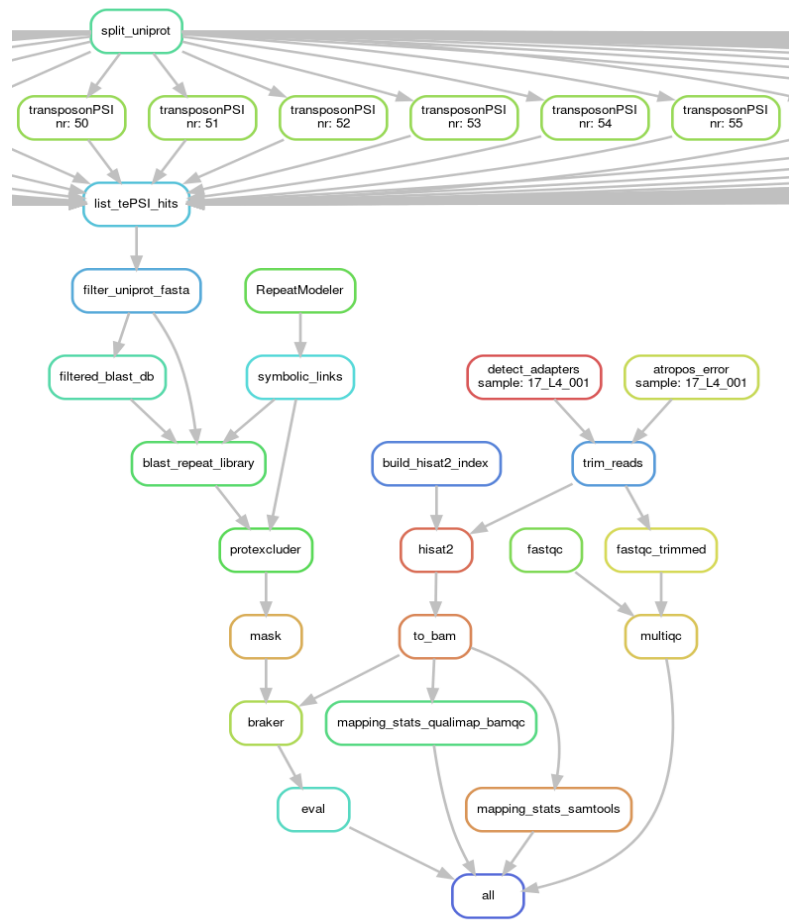
# Pop quiz

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

```
rule rename_file:  
    input:  
        rules.create_file.output  
    output:  
        'results/renamed_file.txt'  
    shell:  
        'mv {input} {output}'
```

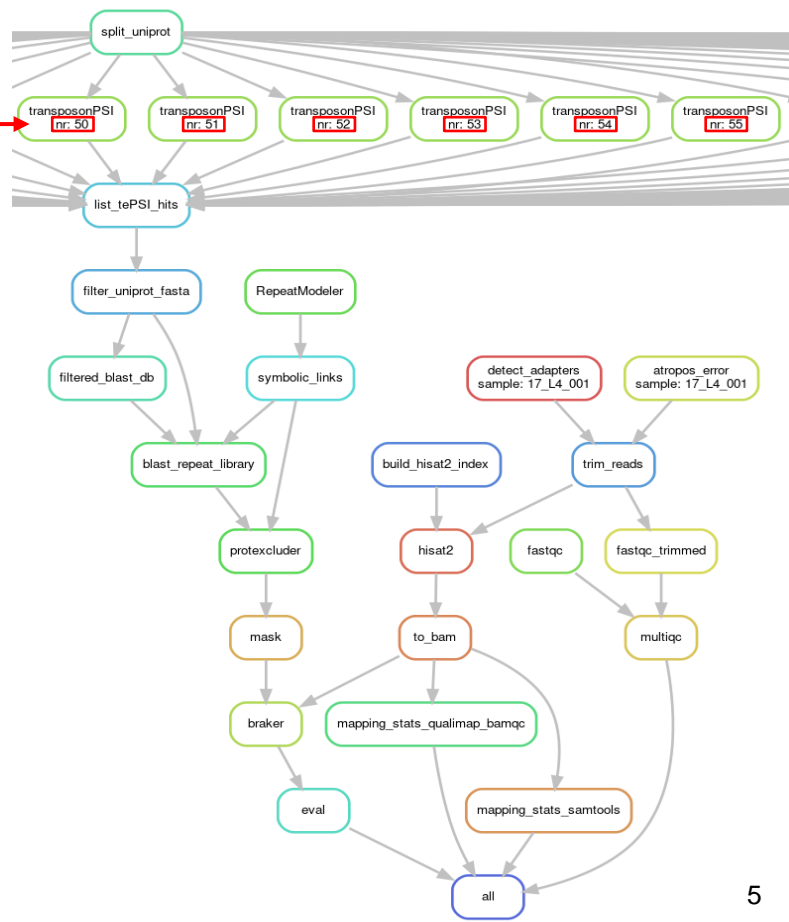
# Building a Directed Acyclic Graph (DAG)

- Snakemake determines which jobs to run to produce desired **outputs**



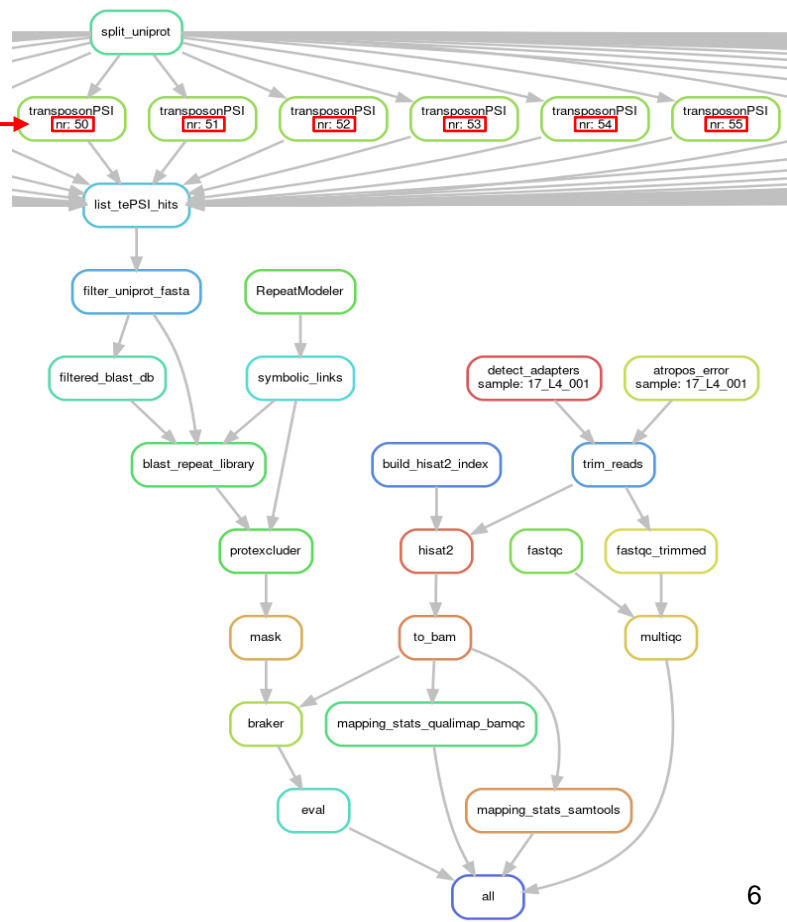
# Building a Directed Acyclic Graph (DAG)

- 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,  $\neq$  flowchart
  - No if/else decisions or loops
  - Snakemake runs every job in the DAG exactly once
- **DAG**  $\neq$  checking **shell** directives
  - Shell commands are tested during execution
    - Works? 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 rename_file:  
    input:  
        'data/test.txt'  
    output:  
        'results/renamed_test.txt'  
    shell:  
        'mv data/test.txt results/renamed_test.txt'
```

# Avoiding hard-coded filepaths: placeholders

```
rule rename_file:  
    input:  
        'data/test.txt'  
    output:  
        'results/renamed_test.txt'  
    shell:  
        'mv data/test.txt results/renamed_test.txt'
```

```
rule rename_file:  
    input:  
        'data/test.txt'  
    output:  
        'results/renamed_test.txt'  
    shell:  
        'mv {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 rename_file:  
    input:  
        'data/test.txt'  
    output:  
        'results/renamed_test.txt'  
    shell:  
        'mv {input} {output}'
```

# Making more general-purpose rules: wildcards

- Wildcards  $\approx$  "variables"  
automatically inferred by  
Snakemake

# Making more general-purpose rules: wildcards

- Wildcards ≈ "variables" automatically inferred by Snakemake

```
rule rename_file:  
    input:  
        'data/test.txt'  
    output:  
        'results/renamed_test.txt'  
    shell:  
        'mv {input} {output}'
```

} Defined paths



# Making more general-purpose rules: wildcards

- **Wildcards** ≈ "variables" automatically inferred by Snakemake

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_test.txt'
    shell:
        'mv {input} {output}'
```

} Defined paths



```
rule rename_file:
    input:
        'data/{file}.tsv'
    output:
        'results/renamed_{file}.txt'
    shell:
        'mv {input} {output}'
```

} Adaptable paths  
with **wildcards**

# Making more general-purpose rules: wildcards

- **Wildcards** ≈ "variables" automatically inferred by Snakemake

- Enclose wildcard name with curly brackets `{ }`

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_test.txt'
    shell:
        'mv {input} {output}'
```

} Defined paths



```
rule rename_file:
    input:
        'data/{file}.tsv'
    output:
        'results/renamed_{file}.txt'
    shell:
        'mv {input} {output}'
```

} Adaptable paths with **wildcards**

# Making more general-purpose rules: wildcards

- **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 cannot contain wildcards**

```
rule rename_file:  
    input:  
        'data/{file}.txt'  
    output:  
        'results/renamed_{file}.txt'  
    shell:  
        'mv {input} {output}'
```

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



```
{file} = "test"
```



```
input: 'data/test.txt'
```

# Making more general-purpose rules: wildcards

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

```
rule rename_file:
    input:
        'data/{file}_{nb}.txt'
    output:
        'results/renamed_{file}_{nb}.txt'
    shell:
        'mv {input} {output}'
```

```
snakemake --cores 1 results/renamed_test_1.txt
```



```
{file} = "test"; {nb} = "1"
```



```
input: 'data/test_1.txt'
```

# Making more general-purpose rules: wildcards

- **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 need to share the same **wildcards**
- **All outputs, logs...** created by a rule must have the same **wildcards!**

```
rule rename_file:
    input:
        'data/{file}.txt'
    output:
        'results/renamed_{file}_{nb}.txt'
    shell:
        'mv {input} {output}'
```

```
snakemake --cores 1 results/renamed_test_1.txt
```



```
{file} = "test"; {nb} = "1"
```



```
input: 'data/test.txt'
```

# Creating rules with multiple inputs/outputs

- Rules can use multiple inputs/outputs

# Creating rules with multiple inputs/outputs

- **Rules** can use multiple **inputs/outputs**
  - **Separated by a comma**
  - **Input** values are **unpacked** (replaced by a space-separated list)

```
rule gather_files:  
    input:  
        'data/test1.txt',  
        'data/test2.txt'  
    output:  
        'results/merged_test.txt'  
    shell:  
        'cat {input} > {output}'
```



```
shell:  
'cat data/test1.txt data/test2.txt > results/merged_test.txt'
```

# Creating rules with multiple inputs/outputs

- **Rules** can use multiple **inputs/outputs**
  - **Separated by a comma**
  - **Input** values are **unpacked** (replaced by a space-separated list)
- **Shell** can have multiple commands
  - **Separated by a semicolon**
  - Commands are concatenated

```
rule gather_files:
    input:
        'data/test1.txt'
        'data/test2.txt'
    output:
        'results/merged_test.txt'
    shell:
        'cat {input} > {output}'
        'cat {input} >> {output}'
```



# Creating rules with multiple inputs/outputs

- **Rules** can use multiple **inputs/outputs**
  - Separated by a comma
  - Input values are **unpacked** (replaced by a space-separated list)
- **Shell** can have multiple commands
  - ~~Separated by a semicolon~~
  - Commands are concatenated

```
rule gather_files:
    input:
        'data/test1.txt'
        'data/test2.txt'
    output:
        'results/merged_test.txt'
    shell:
        '''
        cat {input} > {output}
        cat {input} >> {output}
        '''
```

# Creating rules with multiple inputs/outputs

- **Rules** can use multiple **inputs/outputs**
  - Separated by a comma
  - Input values are **unpacked** (replaced by a space-separated list)
- **Shell** can have multiple commands
  - ~~Separated by a semicolon~~
  - Commands are concatenated
- **Inputs** can be accessed by their positional index: `input[n]`
  - **Numbering starts at 0**

```
rule gather_files:
    input:
        'data/test1.txt'
        'data/test2.txt'
    output:
        'results/merged_test.txt'
    shell:
        '''
        cat {input[0]} > {output}
        cat {input[1]} >> {output}
        '''
```

# Creating rules with multiple inputs/outputs

- **Rules** can use multiple **inputs/outputs**
  - Separated by a comma
  - Input values are **unpacked** (replaced by a space-separated list)
- **Shell** can have multiple commands
  - ~~Separated by a semicolon~~
  - Commands are concatenated
- **Inputs** can be accessed by their positional index: `input[n]`
  - Numbering starts at 0
- Named **inputs** can be accessed by their names: `input.input_name`

```
rule gather_files:
    input:
        file_1='data/test1.txt',
        file_2='data/test2.txt'
    output:
        'results/merged_test.txt'
    shell:
        '''
        cat {input.file_1} > {output}
        cat {input.file_2} >> {output}
        '''
```

# Creating rules with multiple inputs/outputs

- **Outputs** work like **inputs**
  - Separated by ','
  - Can be named
  - Can be accessed by positional index or by name
- All **outputs** need to be created or the job will fail

```
rule gather_files:  
    input:  
        file_1='data/test1.tsv'  
        file_2='data/test2.tsv'  
  
    output:  
        copy_1='results/copied_test1.txt'  
        copy_2='results/copied_test2.txt'  
  
    shell:  
        '''  
  
        cat {input.file_1} > {output.copy_1}  
        cat {input.file_2} > {output.copy_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 one **rule**
  - Can be accessed with a placeholder in **shell**:  
`{log}`

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    log:
        'logs/renaming.log'
    shell:
        'mv {input} {output} 2> {log}'
```

# Checking Snakemake behaviour: log files

- 'log' is a **directive**; its value is a path to a log file for one **rule**
  - Can be accessed with a placeholder in **shell**:  
`{log}`
- You need to **manually redirect messages to logs**, but Snakemake automatically creates the folder path

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    log:
        'logs/renaming.log'
    shell:
        'mv {input} {output} 2> {log}'
```

# Checking Snakemake behaviour: log files

- 'log' is a **directive**; its value is a path to a log file for one **rule**
  - Can be accessed with a placeholder in **shell**:  
`{log}`
- You need to **manually redirect messages to logs**, but Snakemake automatically creates the folder path
- Log files must have the **same wildcards as the output!**
- Good practice: put all logs in same folder

```
rule rename_file:
    input:
        'data/test.txt'
    output:
        'results/renamed_file.txt'
    log:
        'logs/renaming.log'
    shell:
        'mv {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**

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

# 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

