Intro to Workflow management systems

Brice Letcher and Paul Saary

Outline

- Intro into WMS
  - How we write workflows + how snakemake parses them
  - DAG construction

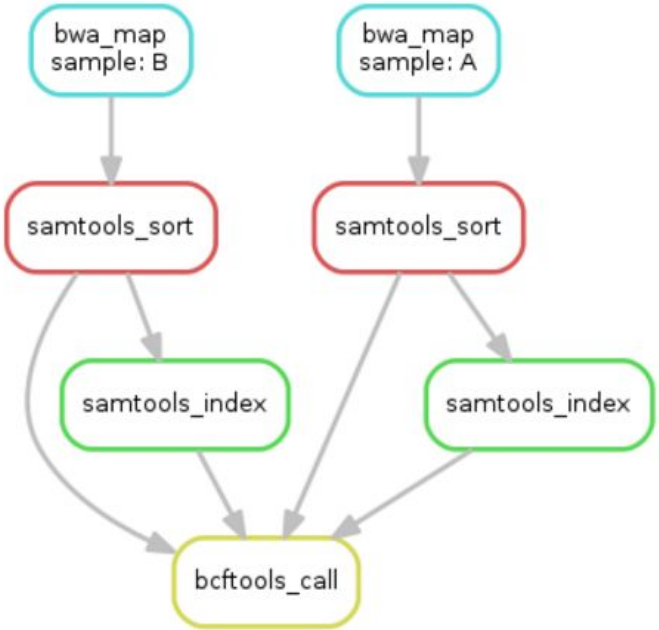
- Snakemake
  - Rules
  - Execution in shell

Why workflow management?

Workflow management handles complicated things like:

- **Forking** processes: running independent processes simultaneously
- **Rejoining** processes: combining the output from independent processes once they have completed
- Setting up process **environments** (eg access to tools, libraries), allocated **resources** (threads, RAM), **logging** to files.
- Creating **reports** showing, for *eg*, the time taken by each process.
- **Deploying** your pipeline on different platforms: Mac/Windows/Linux, different clusters, the cloud.
- **Sharing** your pipeline: readability & how easy it is to modify.
- **Restart** your pipeline where it last failed/stopped.

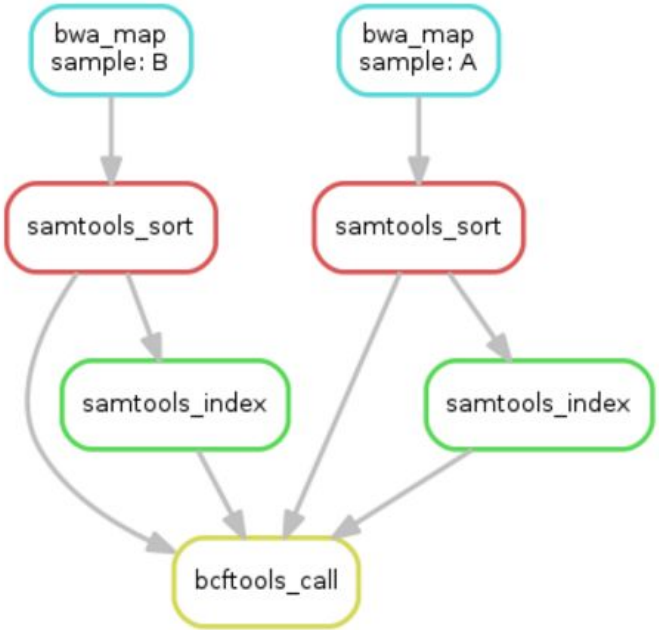
DAG representation



A workflow can be represented as a Directed Acyclic Graph (DAG) where the nodes are processes and a directed edge connects process A to process B if B needs to run only after A has completed. Logically, this is because process B needs data as input which process A can provide!

The graph is acyclic because a process cannot require itself to have run before running.

Snakemake



- Python language (=you can use Python normally) extended with domain specific syntax: **rules**
- DAG nodes are the rules
- Snakemake computes the dependencies (edges) by linking required inputs for one node with produced outputs from another

Rule: Basic syntax

```
rule sort:
    input:
        "path/to/dataset.txt"
    output:
        "dataset.sorted.txt"
    shell:
        "sort {input} > {output}"
```

Snakemake's Rules work on files.

Each rule is a promise: If I find this input file, I will make this output file.

The promise is fulfilled by running the shell code.

Example from:
http://slides.com/johanneskoester

Rules: Wildcards

```
rule sort:
    input:
        "path/to/{dataset}.txt"
    output:
        "{dataset}.sorted.txt"
    shell:
        "sort {input} > {output}"
```

Primary use of wildcards is to run a rule on multiple pieces of data (files).

This way you can generalize a rule and reuse it multiple times.

Example from:
http://slides.com/johanneskoester

Multiple inputs/outputs: access by index

```
rule sort:
    input:
        "path/to/{dataset}.txt"
        "path/to/annotation.txt"
    output:
        "{dataset}.sorted.txt"
    shell:
        "paste <(sort {input[0]}) {input[1]} > {output}"
```

You can have more than one input and more than one output files.

Examplesfrom:
http://slides.com/johanneskoester

Multiple inputs/outputs: access by name

```
rule sort:
    input:
        a="path/to/{dataset}.txt"
        b="path/to/annotation.txt"
    output:
        b ="{dataset}.sorted.txt"
    shell:
        "paste <(sort {input.a}) {input.b} > {output}"
```

It might be easier to name your inputs and outputs.

This way you can keep better track of what is happening.

Example from:
http://slides.com/johanneskoester

Rules: Run python code

```
rule sort:
    input:
        a="path/to/{dataset}.txt"
    output:
        b ="{dataset}.sorted.txt"
    run:
        with open(output.b, “w”) as out:
            for l in sorted(open(input.a)):
                print(l, file=out)
```

Instead of running bash code you can also use Python directly inside the rules run block.

Example from:
http://slides.com/johanneskoester

Rules: Execute a script

```
rule sort:
     input:
          a="path/to/{dataset}.txt"
     output:
          b ="{dataset}.sorted.txt"
     script:
          "scripts/myScript.R"
```

If you give a rule a script to execute, you can access snakemake-related environment variables (eg wildcards) from inside the script.

Python:

```
outputfile = snakemake.output['b']
```

R:

```
outputfile <- snakemake@output$b
```

Example from:
http://slides.com/johanneskoester

Rules: Workflow

```
DATASETS = ["D1", "D2", "D3"]

rule all:
    input:
        expand("{dataset}.sorted.txt", dataset=DATASETS)

rule sort:
    input:
        "path/to/{dataset}.txt"
    output:
        "{dataset}.sorted.txt"
    shell:
        "sort {input} > {output}"rule sort:
```

Native python array

Snakemake needs a ‘target’: a set of outputs that the workflow should produce.

It uses the first rule by default for this.

Example from:
http://slides.com/johanneskoester

Snakefile execution

```
# execute the workflow with target D1.sorted.txt
snakemake D1.sorted.txt

# execute the workflow without target: first rule defines target
snakemake

# dry-run
snakemake -n

# dry-run, print shell commands
snakemake -n -p

# dry-run, print execution reason for each job
snakemake -n -r
```

Examples from:
http://slides.com/johanneskoester

Mental map

How you write: *a* leads to *b* which leads to *c*

→ It can help to write out your DAG before writing the workflow

How Snakemake reads: *c* needs *b*'s output to run which needs *a*'s output to run

→ Seeing this guides writing your rules and debugging

What’s next?

The rest of this workshop is on the webpage:

https://bricoletc.github.io/WMS_teaching

We’ll first walk through a simple workflow in Part I.

Copy paste the file and execute it if you want.