Hands-On-Session Workflow Management

an introduction to **Snakemake**

Caspar Schmitt

LMU,MPP

September 29, 2022

Overview

Motivation

With modern physics analyses being both increasingly

- complex (MVA Analysis, Training Interdependent NNs, ...)
- scaled (more data for e.g. rare processes, precision measurements, ...)

many *undocumented dependencies* emerge.

Motivation

With modern physics analyses being both increasingly

- complex (MVA Analysis, Training Interdependent NNs, ...)
- scaled (more data for e.g. rare processes, precision measurements, ...)

many *undocumented dependencies* emerge.

Manual execution and job steering by analyst is

- error-prone
- time-consuming
- deteriorates the reproducability of results and the transparency of collaborative reviews
- hinders data preservation efforts.

Motivation

With modern physics analyses being both increasingly

- complex (MVA Analysis, Training Interdependent NNs, ...)
- scaled (more data for e.g. rare processes, precision measurements, ...)

many *undocumented dependencies* emerge.

Manual execution and job steering by analyst is

- error-prone
- time-consuming
- deteriorates the reproducability of results and the transparency of collaborative reviews
- hinders data preservation efforts.

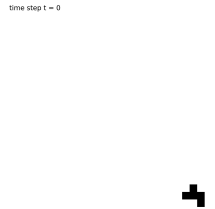
Need Workflow Frameworks!

Focus on **Snakemake** framework which proved most versatile.

Our example workflow: The Game of Life

A **workflow**: a top-level entity of workload to accomplish a scientific objective.
A **task**: a step in a workflow, processes input to produce output.

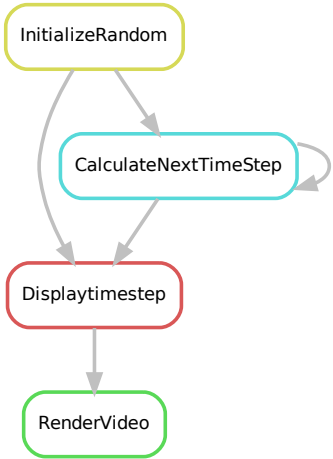
- Simplistic cellular automaton in grid.
- Cells survive or die based on number of neighboring cells.
- Easy to implement in workflow.



Our example workflow: The Game of Life

Exercise 1: Build the Game of Life starting with a random configuration.

Exercise 2: *For advanced users only.* Certain starting grids lead to an empty grid after some time. Make your workflow execution conditional on the current grid configuration and check for grid emptiness.



Directed Acyclic Graph (DAG).

All workflow logic code is centralized in the *Snakefile*.

Workflow tasks are *rules* in the *Snakefile,* and call separate scripts to create output files.

Execute the workflow in the same directory as Snakefile with

```
snakemake --cores 10
```

```
#Snakefile
rule TaskName:
    input:
        "in_file.txt"
    output:
        "out_file.txt"
    params:
        number = 5
    script:
        "python_script.py"
```

```
#in python_script.py
#access filename and
parameters
in_file = snakemake.input
out_file = snakemake.output
par =
snakemake.params.number
#create out_file here
```

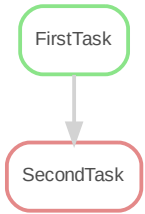
Instead of separate scripts, a task can directly execute shell commands.

```
rule Another_TaskName:
    #may have no or multiple input
    #may have multiple outputs
     output:
         one = "out_file_1.txt",
         two = "out_file_2.txt"
     params:
         number = 5
     shell:
         "echo {params.number} >
{output.one}\
          echo bla > {output.two}"
```

Chaining tasks:

```
rule SecondTask:
    input:
        "intermediate.txt"
    output:
        "final.txt"
    script:
       ”python_script.py”

rule FirstTask:
    output:
        "intermediate.txt"
    shell:
        "echo bla >
{output}"
```

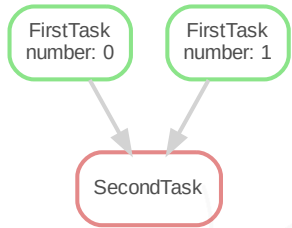
Visualizing tasks in directed acyclic graphs (DAG):



Chaining tasks with wildcards:

```
rule SecondTask:
    input:
        ["intermediate_0.txt",
         "intermediate_1.txt"]
    output:
        "final.txt"
    script:
       ”python_script.py”

rule FirstTask:
    output:
        "intermediate_{number}.txt"
    params:
        par = lambda wildcards:
wildcards.number
    shell:
        "echo {params.par} > {output}"
```

Visualizing tasks in directed acyclic graphs (DAG):



Make your workflow more customizable with configuration files!

```
#Snakefile
configfile: "config.yaml"
rule TaskName:
    input:
        "in_file.txt"
    output:
        "out_file.txt"
    params:
        number = config["p"]
    script:
        "python_script.py"
```

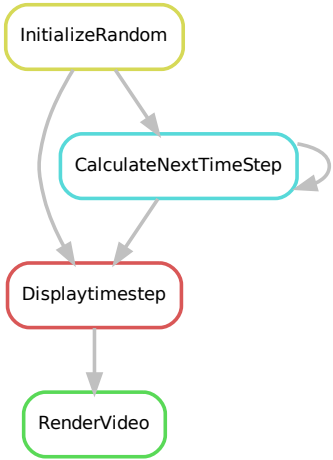
```
#config.yaml

p:10
```

Workflows in **Snakemake**

It's time to try it yourself!

1. Start here.
2. Launch the linked Binder Container, you do not need to install any prerequisites.
3. Start with Exercise 1.
4. Exercise 2 is for advanced users.

You are always welcome to ask anything!



Apropos: Our framework comparison...**your contribution wanted**...

	Criteria	Luigi	Snakemake	Reana
Interface	Workflow language	Python	custom Python-based	Python-based
	Relation to analysis code	integrated	complete factorization	complete factorization
	Boilerplate code	minimal	minimal	significant
	Visualization and monitoring	extensive	no dynamic DAG	no dynamic DAG
	Learning curve	intermediate	simple	simple
Features	Programming languages	single	multiple	multiple
	Data formats	any	any	any
	Dependency management	automatic	automatic	automatic
	Execution control	extensive	extensive	limited
	Error handling and debugging	failed output remains	failed output deleted	single-use clean environment
	Version control	external	external and internal	external and internal
	Scalability	easy	easy	easy
Resources	Environment management	some support	extensive support	some support
	Storage systems support	extensive	extensive	extensive
	Remote execution support	extensive	extensive	extensive
	Authentication mechanism	environment variables	environment variables	access tokens
Install	Installation	pip, non-root	conda, non-root	pip, non-root
	Architecture	single application	single application	server
	State management	target based	target based	target based
	Portability	easy	easy	easy
Support	Documentation	extensive	extensive	incomplete
	Support	extensive	extensive	satisfactory
	System developers	Spotify Group	academic team	CERN
	History and activity	very active	very active	less active
	User community	large	large	significant
	Long term perspective	good	good	acceptable
	Lock-in	yes	no	no
	License	Spotify Group, free use	MIT, free use	CERN
	Use in Punch	Belle 2	LHCb, Radioastronomy	CERN