Introduction to Snakemake

Maria Faria maria.faria@cern.ch

25/11/2024

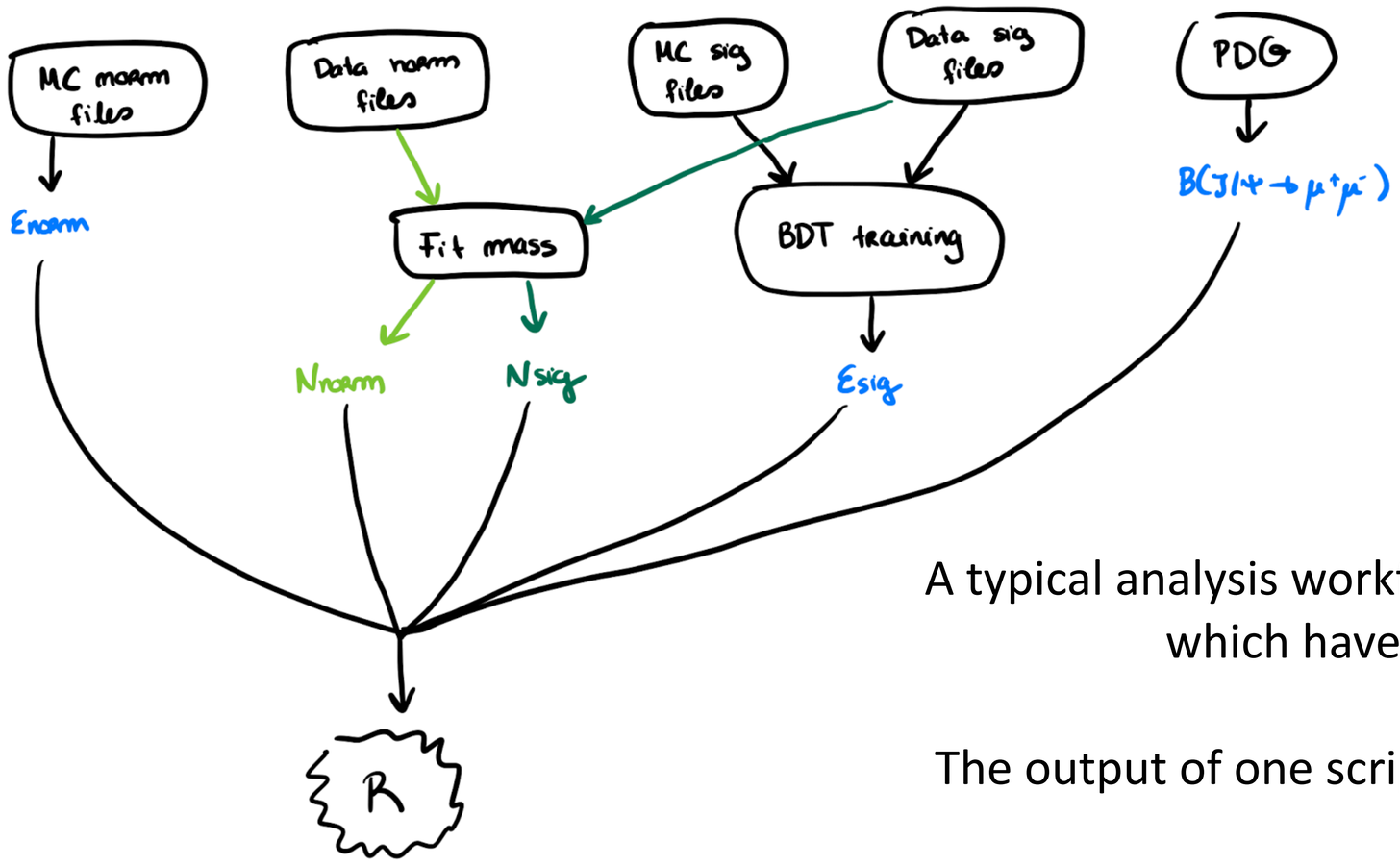
Typical analysis

- Compute the branching fraction of $B^+ \to K^+\mu^+\mu^-$ normalised to the $B^+ \to K^+J/\psi$ channel:

$$R = \frac{\mathscr{B}(B^+ \to K^+\mu^+\mu^-)}{\mathscr{B}(B^+ \to K^+J/\psi)} = \frac{N_{sig}}{N_{norm}} \times \frac{\epsilon_{norm}}{\epsilon_{sig}} \times \mathscr{B}(J/\psi \to \mu^+\mu^-)$$

- Typically computing ϵ_{sig} will involve some BDT training

(Simplified) Analysis workflow



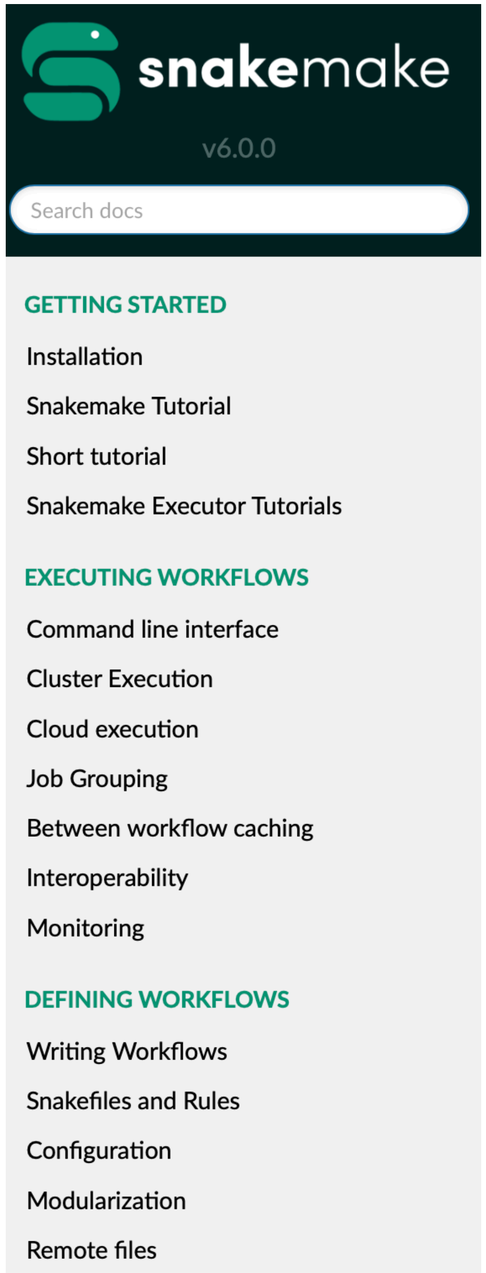
A typical analysis workflow will involve running several scripts, which have dependencies among them

The output of one script can be the input of 1 or more scripts

What if we could obtain R running only a single command? -> **Snakemake**

Introducing snakemake

- Snakemake is “a tool to create reproducible and scalable data”
- It allows us to describe an analysis workflow in a human readable way
- It helps us keeping track of the whole flow of the analysis in a single code
- Useful for us but also to other people who would like to reproduce our results
- It is python based (it compiles pure python)
- The official documentation can be found here
- A basic tutorial can be found in Analysis Essentials



Introducing snakemake

- Snakemake allows you to:

  - keep a record of how your scripts are used and what are their input and output dependencies

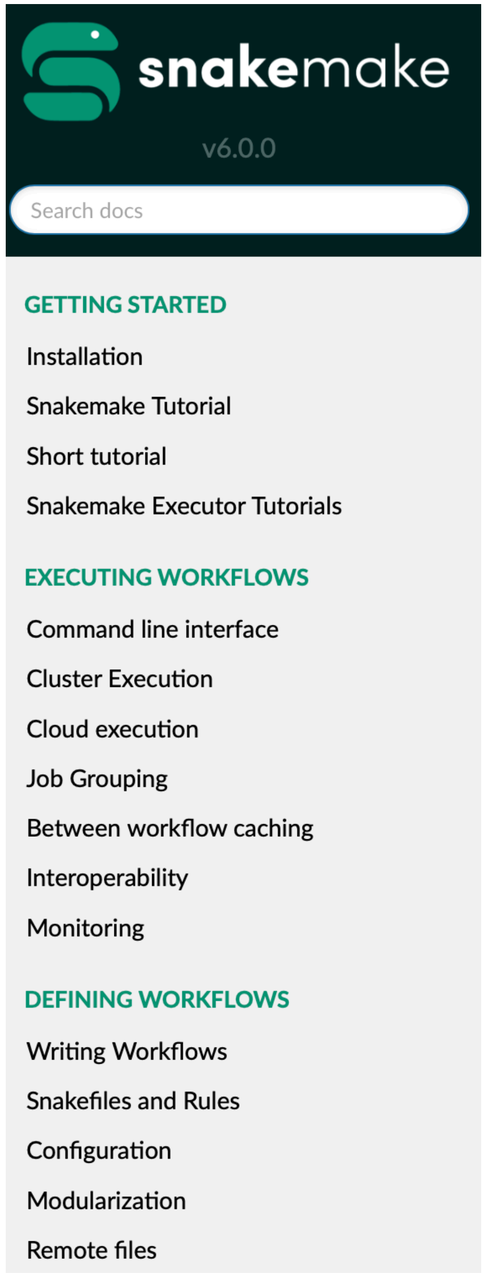
  - run multiple steps in sequence, parallelising when possible

  - automatically detect if something has changed and reprocess data if needed

- Snakemake forces you to:

  - keep your code and paths in order

  - structure your code



Snakefile and rules

- Snakemake workflows are specified with a **snakefile**
- A snakefile contains different **rules**, associated with different scripts
- For each rule we need to specify:

  - input files

  - output files

  - command to run the script

```
rule NAME:
    input: "path/to/inputfile", "path/to/other/inputfile"
    output: "path/to/outputfile", "path/to/another/outputfile"
    shell: "somecommand {input} {output}"
```

Wildcards and expand function

- **Wildcards** can be used to make a rule apply to any number of input files, depending on the value of the wildcard(s)

```
rule complex_conversion:
    input:
        "{dataset}/inputfile"
    output:
        "{dataset}/file.{group}.txt"
    shell:
        "somecommand --group {wildcards.group} < {input} > {output}"
```

- Using the wildcards {dataset} and {group} allows us to run this rule for different values of datasets and groups

- The **expand** function can be used to define the value of the wildcards for which we want to run the rule:

```
rule aggregate:
    input:
        expand("{dataset}/a.{ext}", dataset=DATASETS, ext=FORMATS)
    output:
        "aggregated.txt"
    shell:
        ...
```

Lambda function

- There are other ways in which we can use a wildcard value to specify an input
- Imagine we have a wildcard specifying the sample number and a dataset containing different samples
- The input of our rule is 1 sample from this dataset, depending on the value of the wildcard

```
rule bwa_map:
    input:
        "data/genome.fa",
        lambda wildcards: config["samples"][wildcards.sample]
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"
```

Good practices

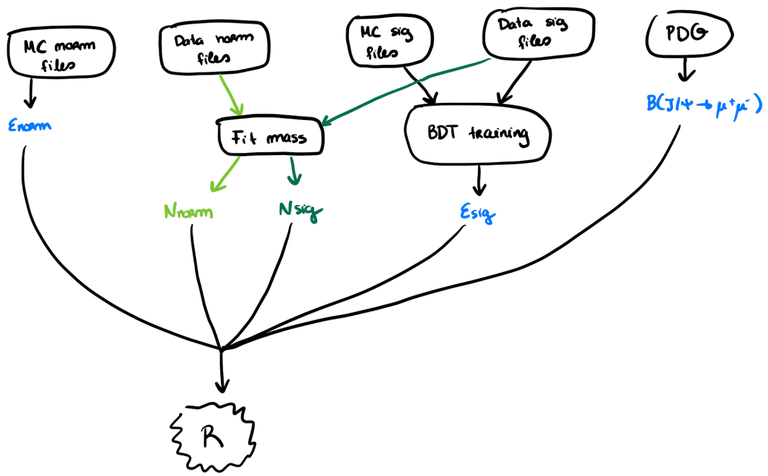
- **Including our script in the input of the rule:** changes in the script will trigger snakemake to re-run the rule and its dependencies

- **Pipe code execution to a log file:** keep track of possible errors during rule execution

```
rule sklearn_training:
    ''' Classifier to seprate signal from combinatorial background '''
    input:
        'Sklearn/make_sklearn_training.py'
    output:
        '/panfs/felician/B2Ktautau/workflow/sklearn_training/DDs/Output_performance_second_step/roc_curve.pdf'
        # '/panfs/felician/B2Ktautau/workflow/sklearn_training/clf_second_step.pkl'
        # '/panfs/felician/B2Ktautau/workflow/sklearn_training/clf_first_step.pkl'
    log:
        '/panfs/felician/B2Ktautau/workflow/sklearn_training/out.log'
    shell:
        'python -u Sklearn/make_sklearn_training.py False 2>&1 | tee {log}'
```

Remember the bash lesson!

Rule all




```
rule all:
  input: expand("{dataset}/file.A.{ext}", dataset=DATASETS, ext=PLOTFORMATS)
```

- <u>At the top of your snakefile</u> you can have a **rule all** where you specify the output you want to obtain from all the rules
- Snakemake will then look over the different rules in your snakefile and figure out which ones need to be run to produce this output

At the end of the day, you can only run a snakemake command to produce the final output of your analysis!

- If you have to change something in an intermediate script, you don't need to figure out which code needs to be run that could be affected by that change

Snakemake will do this for you!

Snakemake example

```
rule all:
    input:
        expand('/panfs/felician/B2Ktautau/workflow/standalone_fitter/201{year}/Species_{species}/{line}.root', year=6, species=2, line=range(1,2368))
```

```
rule run_standalone_fitter:
    ''' Run offline minimisation '''
    input:
        'decay_fit_gsl.py'
    output:
        '/panfs/felician/B2Ktautau/workflow/standalone_fitter/201{year}/Species_{species}/{line}.root'
        # '/panfs/felician/B2Ktautau/workflow/standalone_fitter/201{year}/Species_{species}/Line_{line}/{i_first}.root'
    log:
        '/panfs/felician/B2Ktautau/workflow/standalone_fitter/201{year}/Species_{species}/{line}.log'
        # '/panfs/felician/B2Ktautau/workflow/standalone_fitter/201{year}/Species_{species}/Line_{line}/{i_first}.log'
    resources:
        time = "99:00:00"
    shell:
        # 'root -l -b -q \'DECAY_FIT.C( {wildcards.year}, {wildcards.species}, {wildcards.line} ) \' &> {log};'
        'python -u decay_fit_gsl.py {wildcards.year} {wildcards.species} {wildcards.line} 2>&1 | tee {log};'
```

In the **rule all**, you can specify the value of your wildcards

You can run several instances of your code in parallel

Side note: there are other fancy parameters you can specify in your rule (see more here):

- max amount of time needed (time), amount of memory and disk space needed (mem and disk)…

How to run snakemake (locally)

- **snakemake**: executes the workflow specified in a file ‘snakefile’ in the same directory where the command is run
- **snakemake -s**: allows you to specify the location of your snakefile
- **snakemake -n**: “dry run”, can be used to verify if the workflow is set up correctly and to estimate the amount of computation power needed
- **snakemake -- cores N**: allows you to specify the number of cores used for running; by default snakemake uses the number of available CPU cores in the machine

How to run snakemake (non-locally)

- Snakemake jobs can be run on a cluster work node or cloud infrastructure

- Profiles let snakemake adapt to a particular node environment:

  - **Global profile**: is defined in a system-wise or user-specific configuration directory

  - **Workflow specific profile**: used to provide constraints on custom resources a workflow uses

- **snakemake -- profile name**: would expect a folder called “name” in $HOME/.config/snakemake (linux). (e.g. slurm)

- **snakemake -- profile name --jobs N:** allows us to specify how many jobs we want to run in parallel

Your turn!

Snakemake Lesson LHCb Starterkit 2024

master | snakemake-lesson-lhcb-starterkit-2024 / | History | Find file | Edit | Code

Add python version of code
Maria Carolina Feliciano Faria authored 18 hours ago
7f386e3b

Name	Last commit	Last update
README.md	Update README.md	2 days ago
create_dataset.C	Add file that creates RooDataSet f...	2 days ago
create_dataset.py	Add python version	18 hours ago
fit_mass.C	Add file that makes the mass fit	2 days ago
fit_mass.py	Add python version of code	18 hours ago

Link to gitlab repo

We want to know how many signal events we have from the $B^+ \to \bar{D}^0 D_s^+$ decay in 2016!

For that we perform a fit with RooFit to the B^+ mass in data from which we take the signal yield

Workflow:

1) Create a RooDataSet with the mass variable for both MC and data (create_dataset)

2) Fit the MC RooDataSet (fit_mass)

3) Fit the data RooDataSet fixing some parameters to MC (fit_mass)