Snakemake

Division I , EIB

Andres Tanasijczuk[1] , Pradeep Jasal[2],

1. Université Catholique de Louvain (UCL)
2. University of Barcelona (UB)

UCLouvain

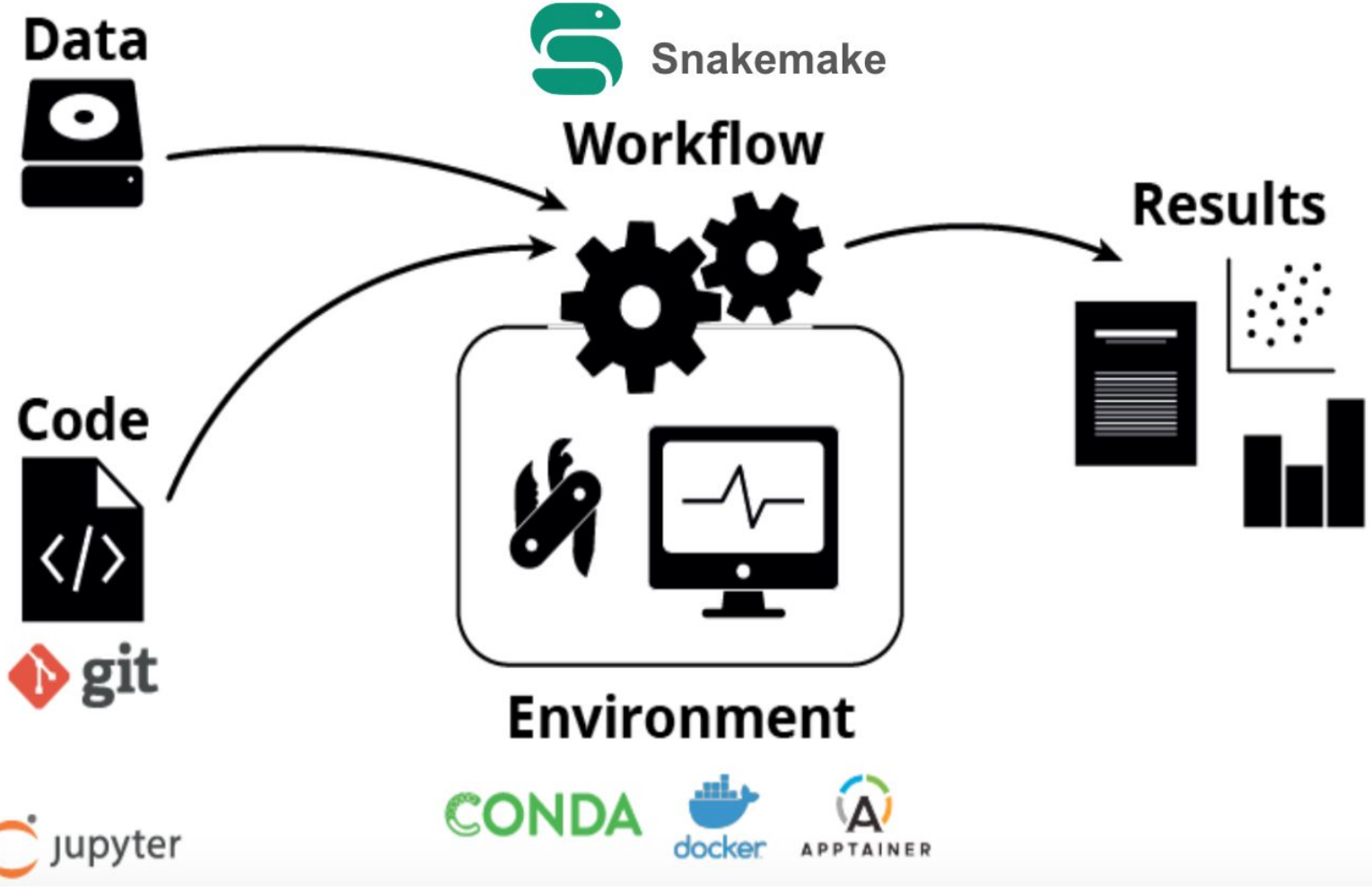
UNIVERSITAT DE BARCELONA

What is covered in this Presentation ?

1. Introduction to Snakemake
    a. What is it ? why use it ? How ?
    b. Running simple example of snakemake
    c. Different options you can use with snakemake command
2. Running pygwb tutorial on local machine
    a. Installing environment with conda
    b. Downloading data from remote server
    c. Run the pipeline locally
    d. Visualize the analysis and rules

Introduction

- Snakemake is a Python based workflow management language
  - Python is very popular in academia
- Developed by bioinformatics community
  - Reproducible
  - Scalable
- Snakemake reads a **Snakefile**
- **Snakefile** defines a set of rules
- rules have inputs and outputs
- Snakemake figures out a correct set of rules to run to produce the output that you want


Data
Snakemake
Workflow
Results
Code
git
Environment
jupyter
CONDA
docker
APPTAINER

Snakefile , Simple Example Tutorial

```
$ tree .
├── input                <------- Input directory
│   ├── address.txt      <----- it contains address
│   └── phone.txt        <----- it contains phone number
└── Snakefile            <------- Snakefile defines workflow and it
                                  contains set of rules that transform
                                  input to output
```

Input data

```
$cat input/address.txt
PersonA : 123 Maple Street, Springfield, IL
PersonB : 456 Oak Avenue, Shelbyville, TN
PersonC : 789 Pine Road, Lakeview, CA
PersonD : 101 Cedar Boulevard, Rivertown, NY
…
```

```
$cat input/phone.txt
PersonA: 00397684512345
PersonB: 00392874561023
PersonC: 00394561238790
PersonD: 00391230785642
…
```

Get information from input data about “PersonP”

```
rule get_info:
    input: ["input/address.txt", "input/phone.txt"]
    output: "output/info.txt"
    shell: "grep PersonP {input[0]} > {output} && grep PersonP {input[1]} >> {output}"


#use –cores option : to assign how many cores to use
$ snakemake --cores 1

$cat output/info.txt
PersonP : 444 Magnolia Avenue, Sunnytown, AZ
PersonP: 00392345678901
```

Get the same information with Parallel workflow “PersonP”

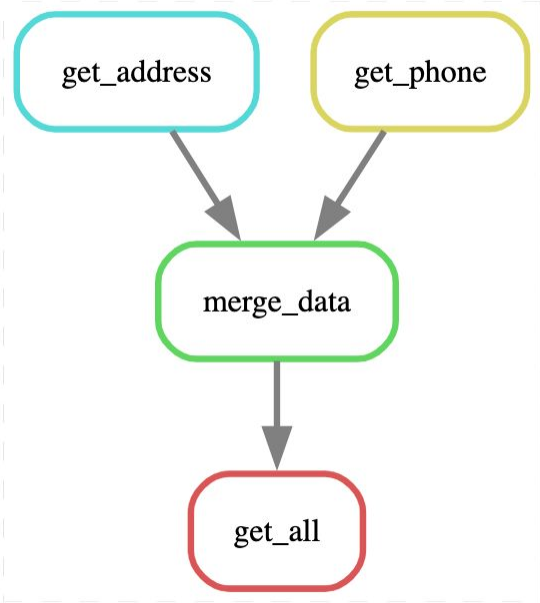
```
rule get_all:
    input: "output/PersonP/info.txt"

rule get_address:
    input: "input/address.txt"
    output: "output/PersonP/address.txt"
    shell: "grep PersonA {input} > {output}"


rule get_phone:
    input: "input/phone.txt"
    output: "output/PersonP/phone.txt"
    shell: "grep PersonA {input} > {output}"

rule merge_data:
    input:
        address="output/PersonP/address.txt",
        phone="output/PersonP/phone.txt",
    output: "output/PersonP/info.txt"
    shell: "cat {input.address} > {output} && cat {input.phone} >> {output}"
```

```
$tree output/
output/
└── PersonP
    ├── address.txt
    ├── info.txt
    └── phone.txt
```

```
$cat output/PersonP/info.txt
PersonP : 444 Magnolia Avenue,
Sunnytown, AZ
PersonP: 00392345678901
```

Generate DAG of workflow

```
$snakemake --dag | dot -Tsvg > dag.png
```



Get the information for multiple Person using {wildcards}

```
rule run_all:
   input: ["output/PersonP/info.txt", "output/PersonQ/info.txt", "output/PersonR/info.txt"]

rule merge_data:
   input:
       address="output/{name}/address.txt",
       phone="output/{name}/phone.txt",
   output: "output/{name}/info.txt"
   shell: "cat {input.address} > {output} && cat {input.phone} >> {output}"


rule get_address:
   input: "input/address.txt"
   output: "output/{name}/address.txt"
   shell: "grep {wildcards.name} {input} > {output}"


rule get_phone:
   input: "input/phone.txt"
   output: "output/{name}/phone.txt"
   shell: "grep {wildcards.name} {input} > {output}"
```

```
$tree output/
output/
├── PersonP
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
├── PersonQ
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
└── PersonR    ├──
address.txt
    ├── info.txt
    └── phone.txt
```

Get the information for multiple Person using {wildcards}

```
rule run_all:
   input: ["output/PersonP/info.txt", "output/PersonQ/info.txt", "output/PersonR/info.txt"]

rule merge_data:
   input:
       address="output/{name}/address.txt",
       phone="output/{name}/phone.txt",
   output: "output/{name}/info.txt"
   shell: "cat {input.address} > {output} && cat {input.phone} >> {output}"

rule get_address:
   input: "input/address.txt"
   output: "output/{name}/address.txt"
   shell: "grep {wildcards.name} {input} > {output}"

rule get_phone:
   input: "input/phone.txt"
   output: "output/{name}/phone.txt"
   shell: "grep {wildcards.name} {input} > {output}"
```

"output/PersonP/address.txt"

```
$tree output/
output/
├── PersonP
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
├── PersonQ
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
└── PersonR    ├──
address.txt
    ├── info.txt
    └── phone.txt
```

Get the information for multiple Person using {wildcards}

```
rule run_all:
   input: ["output/PersonP/info.txt", "output/PersonQ/info.txt", "output/PersonR/info.txt"]

rule merge_data:
   input:
       address="output/{name}/address.txt",
       phone="output/{name}/phone.txt",
   output: "output/{name}/info.txt"
   shell: "cat {input.address} > {output} && cat {input.phone} >> {output}"

rule get_address:
   input: "input/address.txt"
   output: "output/{name}/address.txt"
   shell: "grep {wildcards.name} {input} > {output}"

rule get_phone:
   input: "input/phone.txt"
   output: "output/{name}/phone.txt"
   shell: "grep {wildcards.name} {input} > {output}"
```

"output/PersonP/address.txt"

"output/PersonP/phone.txt"

"output/PersonP/info.txt"

```
$tree output/
output/
├── PersonP
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
├── PersonQ
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
└── PersonR    ├──
address.txt
    ├── info.txt
    └── phone.txt
```

Get the information for multiple Person using {wildcards}

```
rule run_all:
   input: ["output/PersonP/info.txt", "output/PersonQ/info.txt", "output/PersonR/info.txt"]

rule merge_data:
   input:
       address="output/{name}/address.txt",
       phone="output/{name}/phone.txt",
   output: "output/{name}/info.txt"
   shell: "cat {input.address} > {output} && cat {input.phone} >> {output}"

rule get_address:
   input: "input/address.txt"
   output: "output/{name}/address.txt"
   shell: "grep {wildcards.name} {input} > {output}"

rule get_phone:
   input: "input/phone.txt"
   output: "output/{name}/phone.txt"
   shell: "grep {wildcards.name} {input} > {output}"
```

"output/PersonP/address.txt"

"output/PersonP/phone.txt"

"output/PersonP/info.txt"

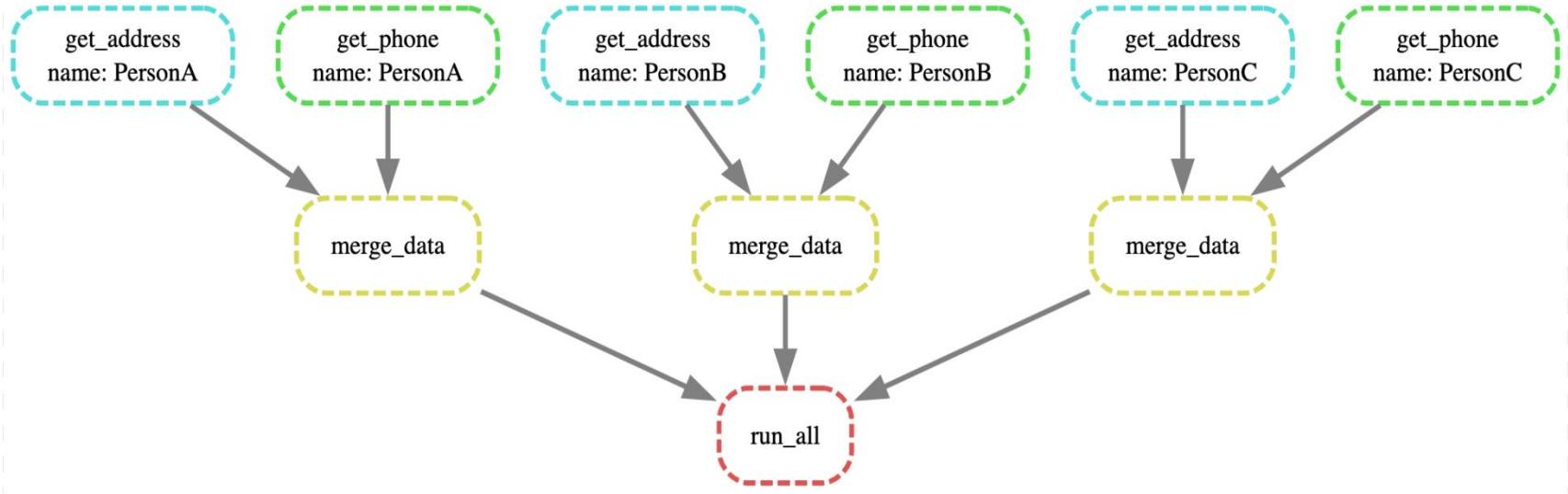
"output/PersonP/address.txt"

"output/PersonP/phone.txt"

```
$tree output/
output/
├── PersonP
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
├── PersonQ
│   ├── address.txt
│   ├── info.txt
│   └── phone.txt
└── PersonR    ├──
address.txt
    ├── info.txt
    └── phone.txt
```

Generating dag for workflow using {wildcard}

```
$snakemake --dag | dot -Tsvg > dag.svg
```



Running pygwb pipeline on local machine

This tutorial is based on the work of Georgy Skorobogatov . Special thanks to Georgy for providing this snakemake pygwb code.

Running pygwb involves following steps

1. Download the input data from origin server
2. generate lalcache files
3. Run pygwb_pipe : it runs cross-correlation stochastic analysis over data from selected detector pair, within the timeframes requested
4. Run pygwb_combine : combines over multiple pygwb_pipe output files
5. Create point_estimate_plots
6. Create zip file out of all results

Setup environment

1. Install conda or mamba from Miniforge : Instructions to follow

```
$ curl -L https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh -o Miniforge3-Linux-x86_64.sh
$ bash Miniforge3-Linux-x86_64.sh
```

2. Git clone the repository

```
$ git clone https://gitlab.et-gw.eu/eib/div1/snakemake-tutorials.git
```

3. Create conda environment : This will create a conda environment “snakemake-tutorial”

```
$ cd snakemake-tutorials/Bologna-2025
$ conda env create -f environment.yaml
```

4. Activate the environment

```
$ conda activate snakemake-tutorial
```

Explore code repository

```
$tree .
├── code
│   └── point_estimate_plots.py
├── config.yaml
├── environment.yaml
├── parameters.ini
├── README.md
├── rucio.cfg
└── Snakefile
```

It contains information about some variable to be used in pygwb pipeline. For example:

```
detectors: ['E1', 'E2']
pygwb_container:"docker://georgysk/pygwb:latest
"
```

file used to create conda environment

It contains parameters to be used in pygwb

Import file in Snakefile

```
# global-scope variables
configfile: "config.yaml"

# range(start, stop, step)
T0s = range(config['TI'],
            config['TF'] - 1,
            config['duration'])
T1s = range(T0s.start + config['duration'],
            T0s.stop + config['duration'],
            config['duration'])

_detectors = sorted(config['detectors'])
_detectors_str =  ''.join(_detectors)
```

`configfile` is immutable Snakemake syntax. It creates global scope `config` object

objects from config.yaml can be accessed as following: `config["objectname"]`

Download input data

```
rule download_data_file_webserver:
        output:
            "data/E-{detector}_STRAIN_"+config['sample']+"-{t0}-2048.gwf"
             # data/E-E1_STRAIN_DATA-1000000000-2048.gwf
             # data/E-E2_STRAIN_DATA-1000000000-2048.gwf
        log:
            "logs/download_data_file_webserver/{detector}_{t0}.log"
        params:
            sample = config['sample'],
            sample_subdir = config['sample'].lower()
        shell:
            "wget -e robots=off -N -r -np -A gwf -P data -nH --cut-dirs=100
http://et-origin.cism.ucl.ac.be/MDC1/v2/{params.sample_subdir}/{wildcards.detector}/E-{wil
dcards.detector}_STRAIN_{params.sample}-{wildcards.t0}-2048.gwf > {log} 2>&1"
```

Download input data

```
rule download_data_file_webserver:
        output:
            "data/E-{detector}_STRAIN_"+config['sample']+"-{t0}-2048.gwf"
             # data/E-E1_STRAIN_DATA-1000000000-2048.gwf
             # data/E-E2_STRAIN_DATA-1000000000-2048.gwf
        log:
            "logs/download_data_file_webserver/{detector}_{t0}.log"
        params:
            sample = config['sample'],
            sample_subdir = config['sample'].lower()
        shell:
            "wget -e robots=off -N -r -np -A gwf -P data -nH --cut-dirs=100
http://et-origin.cism.ucl.ac.be/MDC1/v2/{params.sample_subdir}/{wildcards.detector}/E-{wil
dcards.detector}_STRAIN_{params.sample}-{wildcards.t0}-2048.gwf > {log} 2>&1"
```

Snakemake creates the output directory

{detector} and {t0} are the rule wildcards

convenient way to work with parameters specific to rule

Download input data

```
rule download_data_file_webserver:
        output:
            "data/E-{detector}_STRAIN_"+config['sample']+"-{t0}-2048.gwf"
             # data/E-E1_STRAIN_DATA-1000000000-2048.gwf
             # data/E-E2_STRAIN_DATA-1000000000-2048.gwf
        log:
            "logs/download_data_file_webserver/{detector}_{t0}.log"
       params:
            sample = config['sample'],
            sample_subdir = config['sample'].lower()
        shell:
            "wget -e robots=off -N -r -np -A gwf -P data -nH --cut-dirs=100
http://et-origin.cism.ucl.ac.be/MDC1/v2/{params.sample_subdir}/{wildcards.detector}/E-{wil
dcards.detector}_STRAIN_{params.sample}-{wildcards.t0}-2048.gwf > {log} 2>&1"
```

- Wildcards used in log file name should be same as in output, so to have 1 log file per job
- Logging to log file is not automatic. Here we use "> {log} 2>&1" to redirect stdout/stderr to log file

Producing lalcache files

```
# Rule to create a lal_cache file containing the list of downloaded input data files.

rule lal_cache:
    input:
        expand("data/E-{{detector}}_STRAIN_"+config['sample']+"-{t0}-2048.gwf", t0=T0s)
    output:
        temp("lalcache/{detector}.lcf")
    container:
        config['pygwb_container']
    log:
        "logs/lal_cache/{detector}.log"
    params:
        sample = config['sample']
    shell:
        "/bin/mkdir -p lalcache | "
        "lal_cache data/E-{wildcards.detector}_STRAIN_{params.sample}-*-2048.gwf > {output} 2> {log}"
```

expand function is used to generate list of strings by replacing placeholder(s). Here {{detector}} escapes the placeholder so that {detector} is not substituted and remains a wildcard (i.e. a rule parameter).

```
["data/E-{detector}_STRAIN_DATA-1000000000-2048.gwf",
 "data/E-{detector}_STRAIN_DATA-1000002048-2048.gwf",
 "data/E-{detector}_STRAIN_DATA-1000004096-2048.gwf",
 "data/E-{detector}_STRAIN_DATA-1000006144-2048.gwf",
  .....]
```

pygwb_pipe

```
rule pygwb_pipe:
    input:
        lcfile1 = "lalcache/{detector}.lcf".format(detector=_detectors[0]),
        lcfile2 = "lalcache/{detector}.lcf".format(detector=_detectors[1])
    output:
        "results/parameters_{t0}-{t1}_final.ini",
        "results/point_estimate_sigma_{t0}-{t1}.npz",
        "results/psds_csds_{t0}-{t1}.npz"
    container:
        config['pygwb_container']
    log:
        "logs/pygwb_pipe/{t0}-{t1}.log"
    params:
        detector1 = _detectors[0],
        detector2 = _detectors[1]
    shell:
        "pygwb_pipe --apply_dsc True --calc_pt_est True --pickle_out False --wipe_ifo True --param_file parameters.ini
--t0 {wildcards.t0} --tf {wildcards.t1} --output_path results --interferometer_list {params.detector1}
{params.detector2} --local_data_path {params.detector1}:{input.lcfile1},{params.detector2}:{input.lcfile2} > {log}
2>&1"
```

The inputs for the **pygwb_pipe** rule are the outputs generated by (the two instances of) the previous **lal_cache** rule. The inputs for one instance of the **lal_cache** rule (1 instance = 1 detector) are the outputs of all instances of the **download_data_file_webserver** for a fixed detector (1 instance = 1 pair (segment,detector)). And so on …

pygwb_pipe after replacing placeholders

```
input:
        lcfile1 = "lalcache/E1.lcf"
        lcfile2 = "lalcache/E2.lcf"
.....
output:
        "results/parameters_{t0}-{t1}_final.ini",
        "results/point_estimate_sigma_{t0}-{t1}.npz",
        "results/psds_csds_{t0}-{t1}.npz"
.....
shell:
        "pygwb_pipe --apply_dsc True --calc_pt_est True
--pickle_out False --wipe_ifo True --param_file
parameters.ini --t0 {wildcards.t0} --tf {wildcards.t1}
--output_path results --interferometer_list E1 E2
--local_data_path E1:lalcache/E1.lcf,E2:lalcache/E2.lcf >
{log} 2>&1"
```

1000002048

1000004096

```
For example , let say
["results/parameters_1000002048-1000004096_final.ini",
 "results/point_estimate_sigma_1000002048-1000004096.npz",
 "results/psds_csds_1000002048-1000004096.npz"]
```

{t0} and {t1} substituted from T0s and T1s list.
when snakemake matches a target file like above example, then it extracts values for {t0} and {t1}, and later in the shell command it replaces, {wildcards.t0} and {wildcards.t1} with those values.

pygwb_combine : combines over multiple pygwb_pipe output files

```
rule pygwb_combine:
    input:
        expand("results/point_estimate_sigma_{t0}-{t1}.npz", zip, t0=T0s, t1=T1s),
        expand("results/psds_csds_{t0}-{t1}.npz", zip, t0=T0s, t1=T1s)
    output:
        get_combined_outputs()
    container:
        config['pygwb_container']
    log:
        "logs/pygwb_combine.log"
    params:
        alpha = config['alpha']
    shell:
        "pygwb_combine --data_path results --param_file parameters.ini --alpha {params.alpha} --out_path
results/combined > {log} 2>&1"
```

Create plot from pygwb_combine output

```
rule point_estimate_plots:
    input:
        get_combined_outputs()
    output:
        "results/combined/point_estimate_and_sigma_spectra_up_to_500Hz.png",
        "results/combined/point_estimate_and_sigma_spectra_up_to_200Hz.png"
    container:
        config['pygwb_container']
    log:
        "logs/point_estimate_plots.log"
    shell:
        "python code/point_estimate_plots.py --input_path results/combined --fmax 200 > {log} 2>&1"
```

create zip file of all results

```
rule zip_results:
    input:
        get_combined_outputs(),
        "results/combined/point_estimate_and_sigma_spectra_up_to_500Hz.png",
        "results/combined/point_estimate_and_sigma_spectra_up_to_200Hz.png"
    output:
        "results_combined.zip"
    log:
        "logs/zip_results.log"
    shell:
        "zip -r {output} results/combined > {log} 2>&1"
```

How Snakemake plans its jobs

```
$snakemake --use-singularity --dag | dot -Tpng > dag.png
```

Use -p option with snakemake command, -p print out the shell commands before running them.

```
$snakemake –use-singularity -p
```

```
The -F flag turns on forceall mode and runs all rules
```

```
$snakemake –use-singularity -F
```

Rule	Input	Output
`rule all:`	`"results_combined.zip"`	**None**

`rule zip_results:`	`get_combined_outputs(),`
	`"results/combined/point_estimate_and_sigma_spectra_up_to_500Hz.png",`
	`"results/combined/point_estimate_and_sigma_spectra_up_to_200Hz.png"`
		`"results_combined.zip"`

`rule point_estimate_plots:`	`get_combined_outputs()`	`"results/combined/point_estimate_and_sigma_spectra_up_to_500Hz.png",` `"results/combined/point_estimate_and_sigma_spectra_up_to_200Hz.png"`
`rule pygwb_combine:`	`expand("results/point_estimate_sigma_{t0}-{t1}.npz", zip, t0=T0s, t1=T1s),` `expand("results/psds_csds_{t0}-{t1}.npz", zip, t0=T0s, t1=T1s)`	`get_combined_outputs()`

`rule pygwb_pipe:`	`lalcache/{detector}.lcf`	`"results/parameters_{t0}-{t1}_final.ini",`
		`"results/point_estimate_sigma_{t0}-{t1}.npz",`
		`"results/psds_csds_{t0}-{t1}.npz"`

Rule	Input	Output
`rule lal_cache:`	`expand("data/E-{{detector}}_STRAIN_"+config['sample']+"-{t0}-2048.gwf", t0=T0s)`	`["lalcache/E1.lcf","lalcache/E2.lcf"]`
`rule download_data_file_webserver:`	`None`	`"data/E-{detector}_STRAIN_DATA-{t0}-2048.gwf"`

References

1. Snakemake Documentation
2. HSF HEP Software Foundation Training
3. pygwb documentation
4. LHCb Introduction to Snakemake
5. Software Carpentries

Thank you