FAIR_bioinfo : Open Science and FAIR principles in a bioinformatics project

How to make a bioinformatics project more reproducible

C. Hernandez[1] T. Denecker[2] J. Sellier[2] G. Le Corguillé[2]
C. Toffano-Nioche[1]

[1]Institute for Integrative Biology of the Cell (I2BC)
UMR 9198, Université Paris-Sud, CNRS, CEA
91190 - Gif-sur-Yvette, France

[2]IFB Core Cluster taskforce

June 2021

Schedule

Introduction to snakemake workflow

Schedule

Introduction to snakemake workflow

Exercise 1: one unique step

More on Snakemake

Exercise 2: Running the snakemake workflow on our laptop

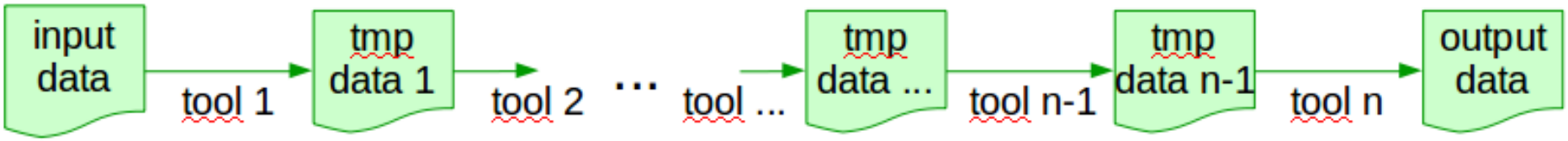
Bonus: From bash script to snakemake

Exercise 3: workflow of the RNAseq analysis

Introduction to snakemake workflow

Workflow definition

a pool of commands, progressively linked by the treatments of the input data towards the results:



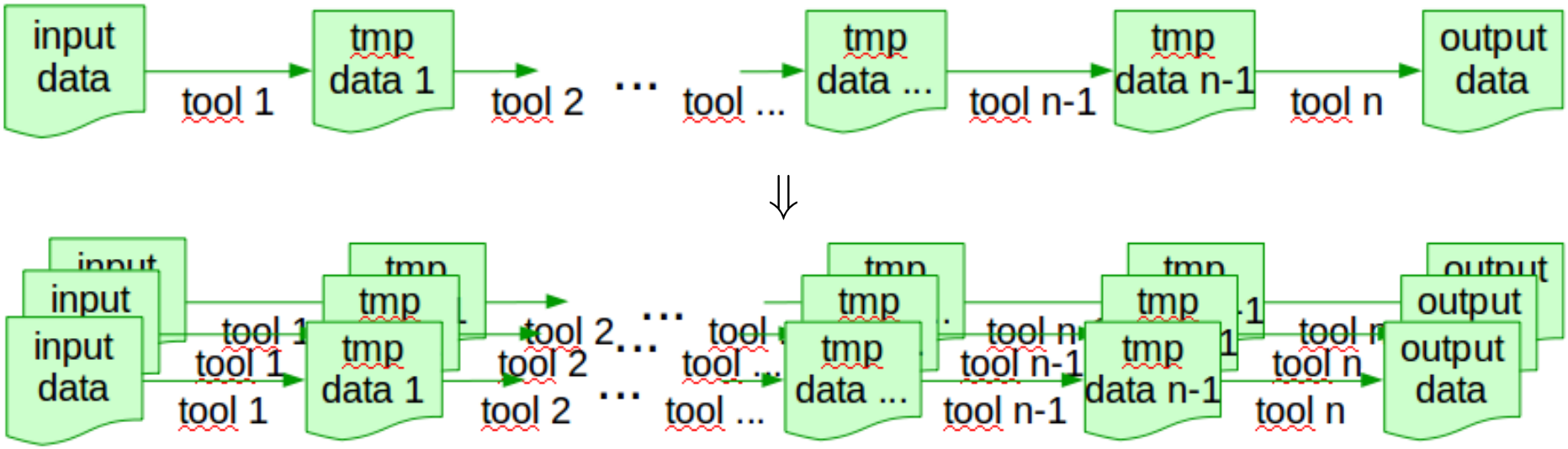
arrow: output of tool $n-1$ = input for tool n

How to save time?

Improve algorithms? Are we ready to optimize Bowtie2? hem ... no!
With multiple data for analysis $\Rightarrow$ we can parallelize.

Data parallelization

Several data flows can be processed in parallel:



With a multi-cores PC or a computational cluster (ex. 2000 cores), we can attribute one core to one workflow.

Workflow management system

Many workflow management systems, many forms:

- command line: shell (but doesn't handle parallelization alone, need to script it, not easy)
- rule:  SNAKEMAKE, CMake, nextflow, ...
- graphic interface: Galaxy, Taverna, Keppler, ...

pros: important for reproducibility (keep track of when each file was generated, and by which operation), manage parallelization
cons: learning effort

We choose  SNAKEMAKE

Snakemake rule

Snakemake: mix of the programming language Python (snake) and the rule-based automation tool Make[1]

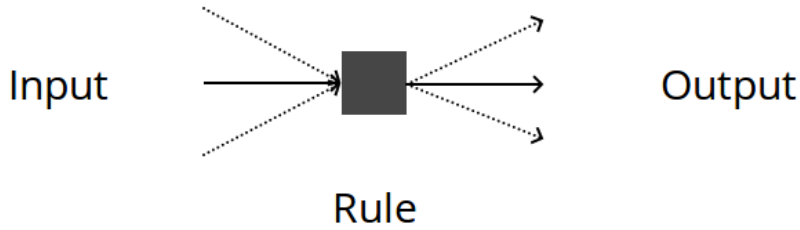
Good practice: one step, one rule

A rule is defined by it name and may contain:

- `input:` list one or more file names
- `output:` list one or more file names
- command (`run:` for python ; `shell:` for shell, R, etc)

+ optional directives: `params:`, `message:`, `log:`, ...

Remark: with 1 command line, use a `shell:` directive ; with many command lines, use a `run:` directive with python shell("...") functions.



[1] Make: https://www.gnu.org/software/make/manual/

Hello World example

The objective of this example is to write "Hello World" into the file `world.txt` in the directory `hello`:

hello_world.smk:

```
1 rule hello_world:
2     output: "hello/world.txt"
3     shell: "echo Hello World > hello/world.txt"
```

- this rule contains only an `output:` directive (`echo` command construction)

Snakemake

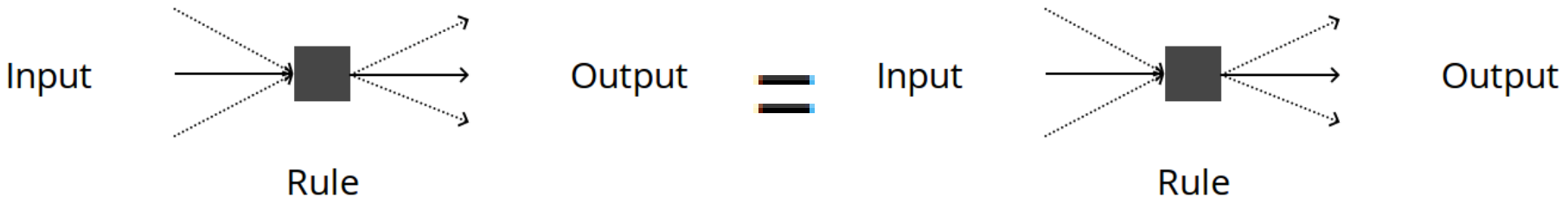
Snakemake automatically makes sure that everything is up to date, otherwise it launch the jobs that need to be.

Snakemake:

- works on files (rather than streams, reading/writing from databases or passing variables in memory)
- is based on Python (but know how to code in Python is not required to work with Snakemake)
- has features for defining the environment with which each task is carried out (running a large number of small third-party tools is current in bioinformatics)
- is easily to be scaled from desktop to server, cluster, grid or cloud environments (ie. develop on laptop using a small subset of data, run the real analysis on a cluster)

Data flow linkage

A snakemake workflow links rules thank to the filenames of the rule input and output directives:

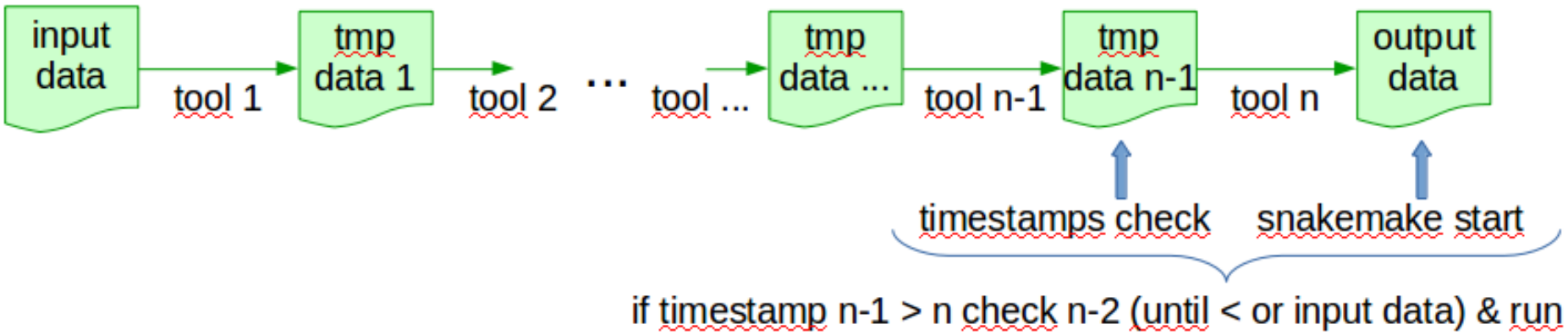


Snakemake rules order:

the first rule (all, target, ...) specifies the result files, the next rules describe how to achieve them.

Rule execution order

Snakemake starts with the first rule that describes the workflow result files. Since output files do not exist, it "goes back" through the workflow until it finds a file to apply a rule to.



For determining whether output files have to be re-created, Snakemake checks whether the file modification date (i.e. the timestamp) of any file is newer than the timestamp of the output file.

Generalization with wilcards

Wildcards (a Snakemake key feature) allow to replace part of filenames:

- reduce hardcoding: more flexible input and output directives, work on new data without modification
- are writing into {}
- are automatically resolved (ie. replaced by regular expression ".+" in filenames)

Wildcards are specific to a rule, a same file can be accessed by different matching:

Ex. with the file "101/file.A.txt"

```
1 rule one: output: "{set}1/file.{grp}.txt" => set=10, grp=A
2 rule two: output: "{set}/file.A.{ext}" => set=101, ext=txt
```

(more on wildcards in the snakemake documentation)

With and without wilcards examples

without_wildcards_uniprot.smk

```
rule all:
 input: "P10415.fasta", "P01308.fasta"

rule get_prot:
 output: "P10415.fasta", "P01308.fasta"
 run:
  shell("wget https://www.uniprot.org/uniprot/P10415.fasta")
  shell("wget https://www.uniprot.org/uniprot/P01308.fasta")
```

with_wildcards_uniprot.smk

```
rule get_prot:
  output: "{prot}.fasta"
  run:
    shell("wget https://www.uniprot.org/uniprot/{wildcards.
    prot}.fasta")
```

Input (output) specifications

enumerated

```
rule one:
  input: "P10415.fasta", "P01308.fasta"
```

python list & wildcards

```
DATASETS = ["P10415", "P01308"]
rule one:
  input: ["{dataset}.fasta".format(dataset=dataset)
          for dataset in DATASETS]
```

expand() & wildcards

```
DATASETS = ["P10415", "P01308"]
rule one:
  input: expand("{dataset}.fasta", dataset=DATASETS)
```

Snakemake accesses

Laptop with docker

```
docker pull snakemake/snakemake  #install (linux: add sudo)
docker run -v ${PWD}:/data -w /data snakemake/snakemake
    snakemake ...  #run
```

Laptop with conda

```
conda create -n smk-env -c bioconda snakemake  #install
conda activate smk-env ; snakemake ...         #run
```

IFB core cluster

```
module load snakemake ; snakemake ...          #run
```

check run: replace `...` by `--version`

Exercise 1 : first snakefile

Practical exercise

For this practical exercise on Snakemake we will:

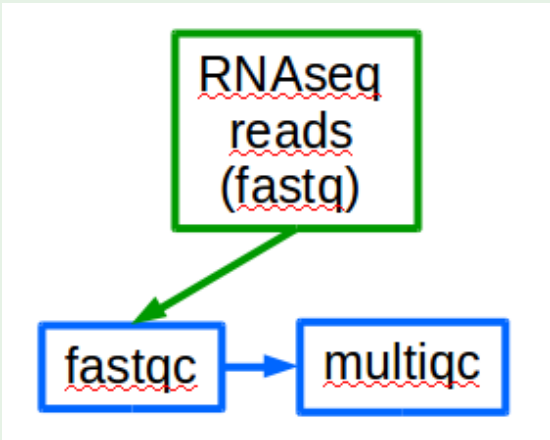
- access to conda by the way of a docker container
- access to snakemake and analysis tools by the way of a conda environment (details about conda will be seen after)
- create a first snakefile with one rule
- add a second rule to create a first workflow

During this first exercise, we will execute several cycles: executing snakemake, observing the result and improving the code. Each code version will be noted `ex1_oX.smk` with X a progressive digit.
(or save on github ...)

Practical exercise

The final objective is to create a snakefile to manage this small workflow:

a small workflow



Input data

The input data, the RNASeq reads files, may be downloaded from: `https://zenodo.org/record/3997237`

Download and unzip

Exercise setup

We will access to the analysis tools thanks to a conda environment, `envfair.yml` (cf. next slide), designed for this small workflow:

Conda environment

```
# do go to the parent directory of Data (from the downloaded
    and unziped data):
cd ......
# create envfair.yml
```

We will access to Snakemake by running a docker image containing the conda tool:

Docker miniconda3

```
docker run -it -v ${PWD}:/data continuumio/miniconda3
cd data
conda env create -n envfair -f envfair.yml
conda activate envfair
```

Exercise setup

envfair.yml

```
channels:
  - conda-forge
  - bioconda
  - default
dependencies:
  # workflow manager:
  - bioconda::snakemake-minimal>=6.5
  # check quality of fastq data (java)
  - bioconda::fastqc=0.11.9
  # R package to aggregate reports
  - bioconda::multiqc=1.9
```

Rule concept with one input file

Objective 1

Create a snakemake file named `ex1_o1.smk` including the first step of the RNAseq workflow (the reads quality checking thank to the `fastqc` tool) on one of the RNAseq files

Hint

- input file: `SRR3099585_chr18.fastq.gz` in a local directory of yours
- fastqc access: by running `docker miniconda3` + activate the conda `envfair` environment
- fastqc command:
  `fastqc inputFileName --outdir FastQCResultDirectory`
- the 2 fastqc result files (`*_fastqc.zip` & `*_fastqc.html`) will be located in the fastqc result directory and will be named based on the prefix of input file (eg. `SRR3099585_chr18_fastqc.zip`)

Solution

ex1_o1.smk

```
rule fastqc:
  output:
    "FastQC/SRR3099585_chr18_fastqc.zip",
    "FastQC/SRR3099585_chr18_fastqc.html"
  input:
    "Data/SRR3099585_chr18.fastq.gz"
  shell: "fastqc --outdir FastQC/ {input}"
```

Snakemake run

```
snakemake --cores 1 --snakefile ex1_o1.smk
```

Observe result

Look at the newly created `FastQC` directory: Snakemake create alone the needed directories.

One rule, 2 input files

Objective 2

Add a second input RNAseq file to the rule

Hint

- input file: `SRR3099586_chr18.fastq.gz` in a local directory of yours
- don't forget the output files

Solution

ex1_o2.smk

```
rule fastqc:
  output:
    "FastQC/SRR3099585_chr18_fastqc.zip",
    "FastQC/SRR3099585_chr18_fastqc.html",
    "FastQC/SRR3099586_chr18_fastqc.zip",
    "FastQC/SRR3099586_chr18_fastqc.html"
  input:
    "Data/SRR3099585_chr18.fastq.gz",
    "Data/SRR3099586_chr18.fastq.gz"
  shell: "fastqc --outdir FastQC/ {input}"
```

Snakemake run

```
snakemake -c1 -s ex1_o2.smk
# -s & -c: short forms of the --snakefile & --cores options
```

Solution

Observe result

Snakemake run the fastqc tool only for the 2nd input file added.

Run again

Run again the snakemake command: `snakemake -c1 -s ex1_o2.smk`
Why does Snakemake reply `"Nothing to be done"`?

Solutions

- delete the FastQC directory (`rm -Rf FastQC`) and rerun the snakemake command
- use the Snakemake `--forcerules` (`-R`) option:
  `snakemake -c1 -s ex1_o2.smk -R fastqc`

Manage all the RNAseq files

Objective 3

Add all the RNAseq files.
Boring with writing all input and output file names?
Use the `expand()` function to manage all the input RNAseq files at once.

Hint

- create a Python list at the begining of the snakefile and containing all the basename of the input files (don't include the "`.fastq.gz`" suffix).
  Python list: `list_name = ["item1", "item2", ..., "itemN"]`
- replace the filename lists of the input and output directives by the `expand()` function

Solution

ex1_o3.smk

```
SAMPLES = ["SRR3099585_chr18","SRR3099586_chr18","
    SRR3099587_chr18"] # add all 6 samples

rule fastqc:
  output:
    expand("FastQC/{sample}_fastqc.zip", sample = SAMPLES),
    expand("FastQC/{sample}_fastqc.html", sample = SAMPLES)
  input:
    expand("Data/{sample}.fastq.gz", sample = SAMPLES)
  shell: "fastqc --outdir FastQC/ {input}"
```

Snakemake run

```
rm -Rf FastQC/
snakemake -c1 -s ex1_o3.smk
```

Add a second rule

Objective 4

Add a second rule: this will start a workflow.
The second tool/rule will aggregate all the fastqc results thank to the R multiqc tool.

Hint

- inputs: the fastqc zip files
- command: `multiqc FastQCResultDirectory`
- 2 outputs: a file `multiqc_report.html` & a repository `multiqc_data`

Solution

ex1_o4.smk (copy, run)

```
SAMPLES = ["SRR3099585_chr18","SRR3099586_chr18","
    SRR3099587_chr18"]

rule fastqc:
  output:
    expand("FastQC/{sample}_fastqc.zip", sample = SAMPLES),
    expand("FastQC/{sample}_fastqc.html", sample = SAMPLES)
  input:
    expand("Data/{sample}.fastq.gz", sample = SAMPLES)
  shell: "fastqc --outdir FastQC/ {input}"

rule multiqc:
  output:
    "multiqc_report.html",
    directory("multiqc_data")
  input:
    expand("FastQC/{sample}_fastqc.zip", sample = SAMPLES)
  shell: "multiqc {input}"
```

Solution

Observe result

Does Snakemake do the job?
Why wasn't the fastqc command launched?

rule links

Snakemake run the first rule (fastqc) and stop when the target files are present.
Solutions ?

- put the multiqc rule before the fastqc rule
- add a rule that aggregate all the rules of the workflow

Adding a new rule is the choice (could be no link between the rules)

The target rule

Objective 5

Add a "first" rule (named "all", "target", ...) with the expected results for all the rules in its `input:` directive.

Solution

ex1_o5.smk

```
...
rule all:
  input:
    expand("FastQC/{sample}_fastqc.html", sample=SAMPLES),
    "multiqc_report.html",
    directory("multiqc_data")
...
```

Snakemake run

```
snakemake -c1 -s ex1_o5.smk -R fastqc
```

Solution

Observe result

Does Snakemake do the job?

Fastqc: job or jobs?

Look at more precisely the fastqc job. We have many input files but snakemake launched only one fastqc job:

```
Job stats:
job         count    min threads    max threads
-------  -------  -------------  -------------
all             1              1              1
fastqc          1              1              1
multiqc         1              1              1
total           3              1              1
```

It is because the `fastqc` rule is defined with a list of files and not for one unique file and because the `fastqc` tool accepts both a unique file as well as a list of files.

Running n individual jobs

Objective 6

Thank to the `all` rule, all expected files are designated. So we don't need to give the `fastqc` rule a list anymore and we can replace it to manage only one file and all files one by one. We will gain in power in systems having more than one core.

Hint

Replace the `expand()` function with a simple wildcard for the filename in the `fastqc` rule.

Solution

ex1_o6.smk

```
rule fastqc:
  output:
    "FastQC/{sample}_fastqc.zip",
    "FastQC/{sample}_fastqc.html"
  input:
    "Data/{sample}.fastq.gz"
  shell: "fastqc --outdir FastQC/ {input}"
```

Snakemake run

```
snakemake -c1 -s ex1_o6.smk -R fastqc
```

Solution

Observe result

Now Snakemake did many fastqc jobs:

```
Job stats:
job         count    min threads    max threads
-------  -------  -------------  -------------
all             1              1              1
fastqc          3              1              1
multiqc         1              1              1
total           5              1              1
```

Parallelize

Rerun with more than one core:

```
1 snakemake -c3 -s ex1_o6.smk -R fastqc
```

What happens now to the runtime displays on the screen?
To correct the mixture, we will move the displays to a log file specific for each rule and each input file.

Adding log file

Objective 7

In Unix systems, the output of a command is usually sent to 2 separate streams: the expected output to Standard Out (stdout, or ">"), and the error messages to Standard Error (stderr, or "2>").
To integrate stderr and stdout into the same log, use "&>". But use it with care because output files are often printed to stdout.

Hint

Redirect the stdout and stderr streams of the `fastqc` and `multiqc` rules by adding a "`log:`" directive with two variables, `out` and `err` to separately redirect each streams.

Solution

ex1_o7.smk

```
# in rule multiqc:
  log:
    out="Logs/multiqc.std",
    err="Logs/multiqc.err"
  shell: "multiqc {input} 1>{log.std} 2>{log.err}"
# in rule fastqc:
  log:
    log1="Logs/{sample}_fastqc.log1",
    log2="Logs/{sample}_fastqc.log2"
  shell: "fastqc --outdir FastQC/ {input} 1>{log.log1} 2>{
    log.log2}"
```

Snakemake run

```
snakemake -c1 -s ex1_o7.smk -R fastqc
```

Little more on Snakemake

Snakemake point

So far, we've seen:

- the rule and the workflow concepts, the snakefile
- how rules are linked thank to input/output files and the first rule, the target rule
- how to generalize the inputs of a rule using wildcards on filenames (and the `expand` function)
- how to redirect `stdout` and `stderr` streams (log)

From now, we will seen some snakemake options:

- adding a configuration file
- getting file names from the file system
- use a conda environment
- to visualize the workflow diagram, use a dry-run option, etc

Using a configuration file

Why use a configuration file?
To place all hard-coding values of the snakefile (paths to files, core numbers, parameter values, etc)

How to?

- create a file written in yml or json (eg. `myConfig.yml`)
- run with the `--configfile myConfig.yml` Snakemake option or ii) add `configfile: myConfig.yml` at the beginning of the snakefile
- in the snakefile, call the defined items with `config["item1"]`

Using a configuration file

myConfig.yml

```
dataDir:
  Data/
```

Replace "Data/..." in inputs by a config call:

```
rule fastqc:
    input: config["dataDir"]+"{sample}.fastq.gz"
```

And Run with the configfile option:

```
snakemake -c1 -s ex1_o7.smk --configfile myConfig.yml
```

File names from the file system

To deduce the identifiers (eg. IDs) of files in a directory, use the inbuilt `glob_wildcards` function:

Eg. of the glob_wilcards function

```
1 IDs, = glob_wildcards("dirpath/{id}.fastq")
```

`glob_wildcards()` matches the given pattern against the files present in the file system and thereby infers the values for all wildcards in the pattern (`{id}` here).

Don't forget the coma after the name (left hand side, `IDs` here).

Conda environment

Snakemake and conda

In the practical exercise we will have one conda environment for executing the whole Snakemake workflow.
Snakemake also supports using explicit conda environments on a per-rule basis:

- add a `conda:` directive in the rule definition :

```
1     conda: rule-specific-env.yml
```

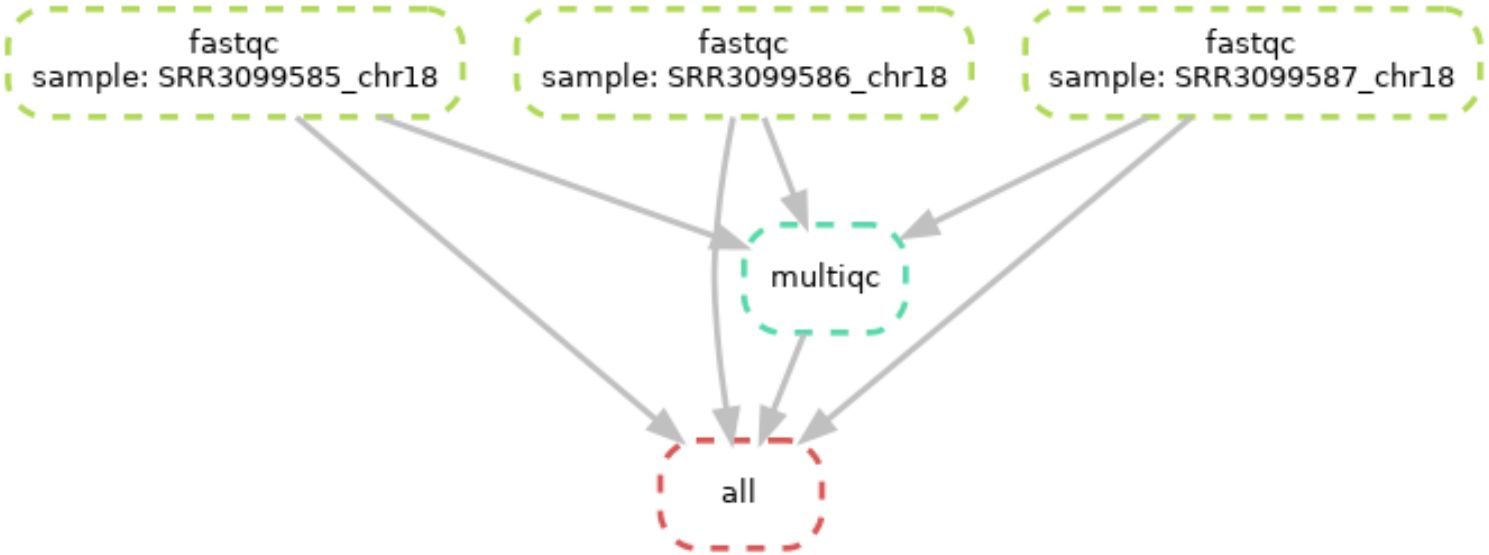
- run Snakemake with the `--use-conda` option

The specified environment will be created and activated on the fly by Snakemake and the rule will then be run in the conda environment.

Snakemake DAG visualization

Snakemake uses `dot` tool (from graphviz package) to create diagrams of the complete workflow (`--dag`) or the rules dependencies (`--rulegraph`):

```
snakemake --dag -s ex1_o7.smk | dot -Tpng > ex1_o7_dag.png
snakemake --rulegraph -s ex1_o7.smk | dot -Tpng >
    ex1_o7_rule.png
```



Other useful options

Running options

- dry-run, do not execute anything, display what would be done: `-n --dryrun`
- print the shell command: `-p --printshellcmds`
- print the reason for each rule execution: `-r --reason`
- print a summary and status of rule: `-D`
- limit the number of jobs in parallel: `-j 1` (cores: `-c 1`)
- automatically create HTML reports (`--report report.html`) containing runtime statistics, a visualization of the workflow topology, used software and data provenance information (need to add the `jinja2` package as a dependency)

all Snakemake options

Last challenge

Clean, delete and re-run !

We may saved the last version of the snakefile and the config file, clean all (but the data) and re-run the workflow.

```
cp ex?_o?.smk RNAseq_analysis.smk
cp ex?_o?.yml RNAseq_analysis_smkEnv.yml
rm -Rf FastQC/ Results/ Logs/ Tmp/ multiqc_*
snakemake -c 1 -s  RNAseq_analysis.smk --configfile
    RNAseq_analysis_smkEnv.yml
```

Snakemake conclusion

Now you can transpose/write any shell script to a snakefile and associate it to a configuration file.

Power gain

- This 2-files solution (snake & config files) will be more powerful when you apply it in a High Performance Computing environment (like the IFB cluster) if you arrange to put all paths in the config file
- I tune up my snakefile with a reduced dataset (typically the first 10000 reads of each input fastq file) before running the full analysis
- For analysis with many data files Snakemake handles error recovery from unintentional interruptions for us: just rerun the snakemake command until each file is processed

Reprodicibility issue

In terms of reproducibility, we have to focus on the tools environment

Ressources

Official documentation https://snakemake.readthedocs.io/en/stable/

Johannes Koëster publication
https://doi.org/10.1093/bioinformatics/bts480

bioinfo-fr.net https://bioinfo-fr.net (+search snakemake)

begining of a gitbook https://endrebak.gitbooks.io/the-snakemake-book