FAIR_bioinfo for bioinformaticians

Introduction to the tools of reproducibility in bioinformatics

C. Hernandez[1] T. Denecker[1] J.Sellier[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]Institut Français de Bioinformatique
à compléter

Sept. 2020

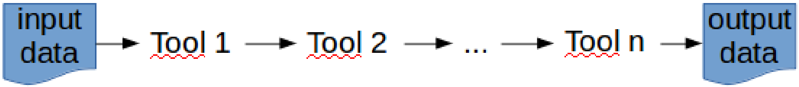
Schedule

Introduction to snakemake workflow

Workflow definition

Linked commands

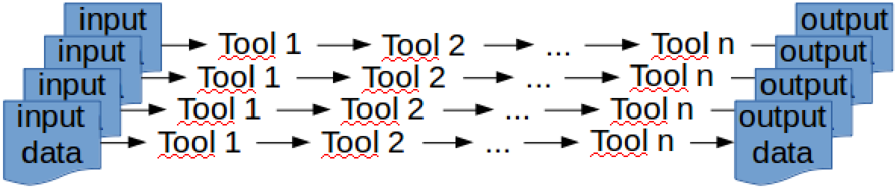
a pool of commands, progressively linked by the treatments of the input data towards the results:



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

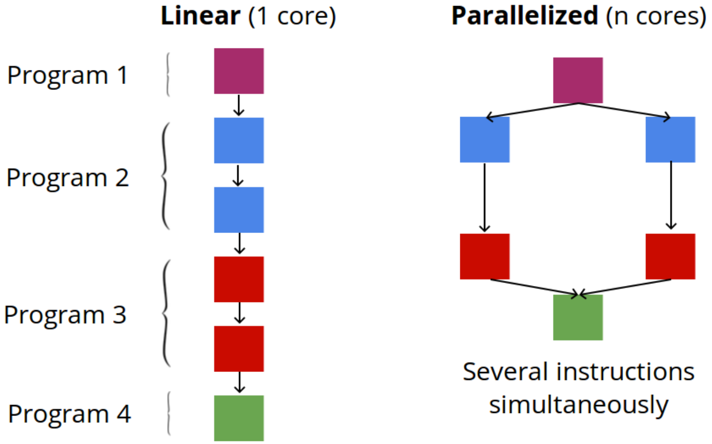
Data parallelization

several data flows can be processed in parallel



A workflow is launched. How to reduce the waiting time?

Improve algorithms? Are we ready to optimize Bowtie2? hem ... no!
We have multiple data and steps of analyses ⇒ we can parallelize!


Linear (1 core)
Program 1
Program 2
Program 3
Program 4
Parallelized (n cores)
Several instructions simultaneously

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]

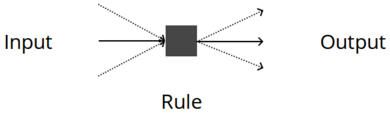
Good practice: one step, one rule

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:

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

- the rule contains only an `output:` directive (due to the usage of the `echo` command)

Snakemake

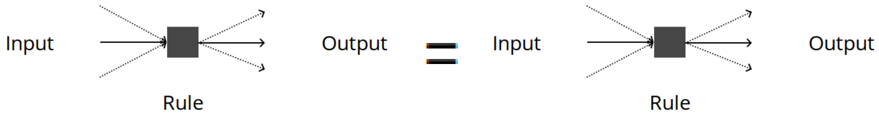
Snakemake automatically makes sure that everything is up to date, otherwise it launch the jobs that need to be.

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:

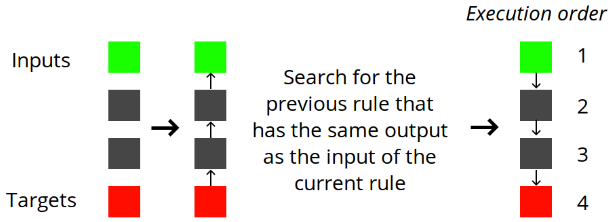


Snakemake rules order:

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 they do not exist, it "goes back" through the workflow until it finds an input file to apply a rule to.


Execution order
Inputs
Targets
Search for the previous rule that has the same output as the input of the current rule
1
2
3
4

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

Generalization with wilcards

Wildcards (Snakemake key feature) allow to replace hardcoded filenames and make input and output directives flexible. Using them will:

- reduce the amount of code needed
- have the workflow work on new data, without modification

In the filename, wildcards (writing into {}) are automatically resolved (replaced by regular expression ".+"). All filenames matching the expression are concerning by the directive. Wildcards are specific to the rule, a same file can be accessed by different matching. :

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

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

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 access by replacing `...` by `--version`

Exercise 1 : first snakefile

Practical exercise

For this practical exercise on Snakemake we will:

- access to snakemake by the way of a dockerfile
- access to 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 the 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.

Exercise setup

We will access to Snakemake by running a docker image containing the conda tool (among other):

docker miniconda3 (git 2.20.1 + conda 4.8.2)

```
docker run -i -t -v ${PWD}:/data continuumio/miniconda3
```

or

docker from NBIS courses (conda 4.6.14 + git 2.7.4)

```
sudo docker run -it -p 8888:8888 -v ${PWD}:/course/
    scilifelablts/reproducible_research_course_slim
```

Conda environment

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

```
conda env create -n envfair -f envfair.yml
conda activate envfair
```

Exercise setup

envfair.yml

```
channels:
  - conda-forge
  - bioconda
  - main
  - default
dependencies:
  - python=3.7.6 # specify python version (not required but
    can help with downstream conflicts)
  - snakemake-minimal=5.10.0 # workflow manager
  - graphviz=2.42.3 # for visualisation
  - xorg-libxrender
  - xorg-libxpm
  - wget=1.20.1 # for downloading files
  - fastqc=0.11.9 # for the RNAseq analysis
  - bowtie2=2.4.1
  - samtools=1.10
  - subread=2.0.1
```

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: running `docker miniconda3` + activate the conda `envfair` environment with
- fastqc command:
  `fastqc inputFileName --outdir ResultDirectory`
- The 2 fastqc result files (.zip & .html) are named based on the prefix of input file

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 --snakefile ex1_o1.smk
```

Observe result

Look at the newly created `FastQC` directory: Snakemake create 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

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

```
# -s is the short form of the --snakefile option
snakemake -s ex1_o2.smk
```

Solution

Observe result

Why does Snakemake reply `"Nothing to be done"`?
Two solutions:

- delete the FastQC directory (`rm -Rf FastQC`) and rerun the snakemake command
- use the Snakemake `--forcerules` (`-R`) option:
  `snakemake -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 others 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 -s ex1_o3.smk
```

Add a second rule

Objective 4

Add a second rule, this will start a workflow.
The second rule concerns the creation of an index file for the genome sequence (needed for the mapping step). As the mapping tool is bowtie2, the index creation tool is bowtie2-build.

Hint

- genome file (input): `Data/O.tauri_genome.fna`
- use a Python list and the `expand()` function to manage the 6 index files names that will be created by the command: "*.1.bt2" ... "*.4.bt2","*.rev.1.bt2","*.rev.2.bt2"
- command:

```
bowtie2-build genomeSequenceAccess indexAccessPrefix
```

Solution

ex1_o4.smk (copy, run)

```
SAMPLES = ["SRR3099585_chr18","SRR3099586_chr18","
    SRR3099587_chr18"]
BIDX = ["1","2","3","4","rev.1","rev.2"]

rule genome_bwt2_index:
  output:
    expand("Tmp/Otauri.{ext}.bt2", ext=BIDX)
  input:
    "Data/O.tauri_genome.fna"
  shell: "bowtie2-build {input} Tmp/Otauri"

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}"
```

Solution

Observe result

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

rule links

Snakemake run the first rule and stop when the target files are present.
Also, there is no link between the 2 rules because they concern two independent parts of the analysis.
The solution is to add a rule that aggregate this 2 parts of the workflow.

The target rule

Objective 5

Add a "first" rule (rule all, target, ...) with the expected results for the 2 rules (`fastqc` and `genome_bwt2_index` in its `input:` directive.

Solution

ex1_o5.smk

```
...

rule all:
  input:
    expand("FastQC/{sample}_fastqc.html", sample=SAMPLES),
    expand("Tmp/Otauri.{ext}.bt2", ext=BIDX)

...
```

Snakemake run

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

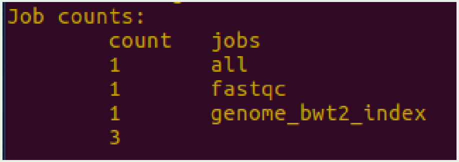
Solution

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 counts:
        count   jobs
        1       all
        1       fastqc
        1       genome_bwt2_index
        3
```

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 wildcard for one 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}"
```

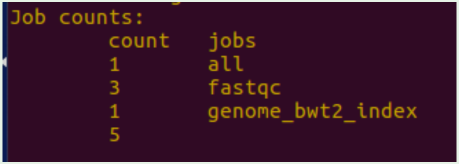
Snakemake run

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

Solution

Observe result

Now Snakemake did many fastqc jobs:



```
Job counts:
        count   jobs
        1       all
        3       fastqc
        1       genome_bwt2_index
        5
```

But what happens to the runtime displays on the screen?
To correct this, 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 two separate streams: the normal output: to Standard Out (stdout also ">" in shell), and error messages: to Standard Error (stderr, or "2>" in shell). To integrate stderr into the same log file as the stdout can be use "&>" instead of ">":
shell: ... `&> {log}`", but use with care when output files are printed to stdout (as often in shell comands).
Redirect the stdout and stderr streams of the `fastqc` and bowtie2-build commands.

Hint

For the `bowtie2-build` and `fastqc` rules, add the `log:` directive with two variables (`log1` and `log2`) to redirect each streams.

Solution

ex1_o7.smk

```
# in rule genome_bwt2_index:
  log:
    log1="Logs/genome_bwt2_index.log1",
    log2="Logs/genome_bwt2_index.log2"
  shell: "bowtie2-build {input} Tmp/Otauri 1>{log.log1} 2>{
    log.log2}"
# 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

```
rm -Rf FastQC/ Tmp/ Logs/; snakemake -s ex1_o7.smk
```

How To?

From bash script to snakemake workflow

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)

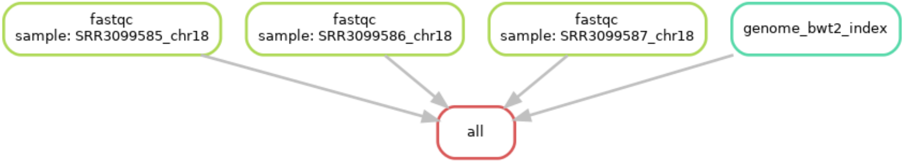
From now, we will seen:

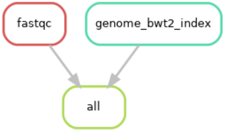
- some snakemake options: to visualize the workflow diagram, use a dry-run option, etc
- adding a configuration file
- getting file names from the file system
- the container directive ???
- how to run snakemake on cluster ???

Snakemake DAG visualization

Snakemake use Graphviz (dot command) to create graphical visualisations, for the complete workflow (--dag) or for the rules dependencies (--rulegraph):

```
snakemake --dag | dot | display
snakemake --dag | dot -Tpng > dag.png
```





Some Snakemake options

Running options

- 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 jinja2 in dependancies)
- dry-run, do not execute anything, display what would be done: `-n --dryrun`
- print the executed 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: `--cores 1`)

all Snakemake options

Some Snakemake options: configuration file

Why and how to use a configuration file?

To place all hard-coding values of the snakefile (paths, core numbers, parameters, etc)

- file: can be write both in yml or in json
- run: add Snakemake option: `--configfile file.yml` or add `configfile: file.yml` at the beginning of the snakefile
- writing: snakefile call of defined items: `config["item1"]` in input/output directive and `{config[item1]}` in shell directive

File names from the file system

To infer the identifiers (IDs) from present files in a directory, use the inbuilt `glob_wildcards` function:

Ex. of the glob_wilcards function

```
IDs, = glob_wildcards("thedir/{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

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 the `conda: rule-specific-env.yml` directive in the rule definition
- 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.

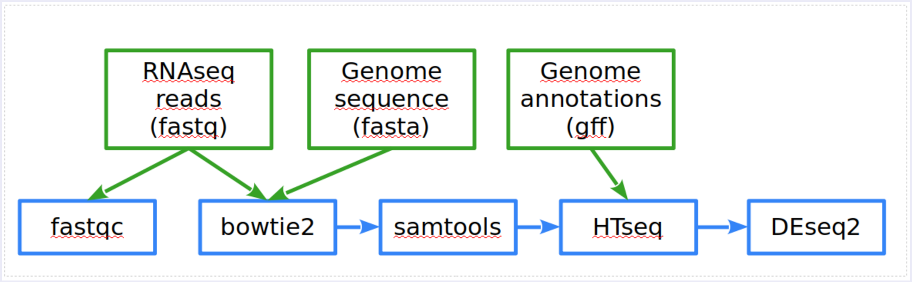
Container directive

Cluster option

Exercise 2: workflow of the RNAseq analysis

RNAseq analysis

Analysis workflow



green=input, blue=tool

fastqc	control quality of the input reads
bowtie2	reads mapping on the genome sequence
samtools	mapped reads selection & formatting
HTseq	count table of mapped reads on genes (annotations)
DEseq2	statistical analysis: genes list having differential expression

Data and command line

Data

-g genome sequence acces (including extention .fna, .fasta)
-a genome annotation acces (inluding extention .gff)
-d RNAseq sample prefix
next args: RNAseq sample prefix, no .fastq.gz extention

Bash command line

```
FAIR_initial_script.sh -g ../O.tauri_genome.fna -a ../O.
    tauri_annotation.gff -d ../ SRR3099585_chr18 S*86_chr18
    S*87_chr18 S*97_chr18 S*98_chr18 S*99_chr18
```

Script in 3 main blocks

```
1) while getops do ... done
2) for sample in $* ; do ... done
3) creation of the result file, counts.txt, with paste, awk,
    and sed bash commands
```

Complete bash script, 1/3

getops block

```
while getopts g:a:d: flag do
        case $flag in
            g)  genome=$OPTARG
                echo genome is $genome ;;
            a)  annots=$OPTARG
                echo annotation is $annots ;;
            d)  rnadir=$OPTARG
                echo RNAseq path is $rnadir ;;
            :)  echo "L'option $OPTARG requiert un argument"
                exit 1 ;;
            \?) echo "$OPTARG : option invalide"
                exit 1 ;;
        esac
   done
shift $(( OPTIND - 1 ))  # shift past the last flag or
    argument
echo samples are $*
```

Complete bash script, 2/3

for block

```
nbs=0;
for sample in $* ; do
   nbs=$(expr ${nbs} + 1)
   echo traitement of sample ${sample}
   # --------- quality control of reads
   if [ ! -d FastQC ]; then
       mkdir FastQC
   fi
   fastqc --outdir FastQC ${rnadir}${sample}.fastq.gz >
    FastQC/${sample}.log 2>&1
   #---------- reads mapping
   if [ ! -d Bwt2_index ]; then
       mkdir Bwt2_index
       bowtie2-build ${genome} Bwt2_index/tauri > Bwt2_index
    /Bwt2_index.log 2>&1
   fi
```

Complete bash script, 3/3

for block, continuation

```
    bowtie2 -x Bwt2_index/tauri -U ${rnadir}${sample}.fastq.gz -S ${sample}.sam > ${sample}_bowtie2.log 2>&1
    #---------- selection and format modification
    samtools view -b ${sample}.sam -o ${sample}.bam
    samtools sort ${sample}.bam -o ${sample}_sort.bam
    samtools index ${sample}_sort.bam
    #---------- counting of mapped reads by gene
    featureCounts -t gene -g ID -a ${annots} -s 1 -o ${sample}_ftc.txt ${sample}_sort.bam > ${sample}_ftc.log 2>&1
done
```

Count table block

```
paste *_ftc.txt > ftc_tmp.txt
awk -v nb=${nbs} -v col=7 'BEGIN{FS="\t"}{ctmp=$1; for(i=col;i<=nb*col;i=i+col){count=sprintf("%s\t%s",ctmp,$i);ctmp=count};print count}' ftc_tmp.txt | sed 1d > counts.txt
```

Exercise 2

Continue the snakefile of the previous exercise in order to replace the bash script.
We will:

Objectives

- add a configuration file
- use a builtin snakemake function to get filenames of the input RNAseq data
- add rules to replace the mapping, formatting, counting, and counts aggregating steps of the bash script

ex2_o1.smk

```
cp ex1_o7.smk ex2_o1.smk
```

getops block

Shell script

```
while getopts g:a:d: flag do
        case $flag in
            g)  genome=$OPTARG
            ...
```

We will use a configuration file:

Objective 1

Add a configuration file, named `RNAseq.yml`, containing both the genome sequence and the annotation files manes, and the access to the Data directory.
In the snakefile, change the configured variables (ex. replace `Data/` and genome by their `config[]` values). The Python strings concatenation is +
Then, run snakemake with the `--configfile` option.

Adding a configuration file

ex2_o1.yml

```
genome:
  O.tauri.fna
annots:
  O.tauri.gff
dataDir:
  Data/
```

ex2_o1.smk: "Data/..." in inputs replaced by a config call:

```
rule genome_bwt2_index:   config["dataDir"]+config["genome"]
rule fastqc:   config["dataDir"]+"{sample}.fastq.gz"
```

snakemake run:

```
rm -Rf FastQC/ Result/ Tmp/ Logs/ ; snakemake -s ex2_o1.smk
    --configfile ex2_o1.yml
```

for block

Shell script

```
nbs=0;
for sample in $* ; do
    ...
done
```

To manage all *.fastq.gz files in a directory, use the `glob_wilcards()` function. In `ex2_o2.smk`, replace the SAMPLES definition by:

ex2_o2.smk

```
SAMPLES, = glob_wildcards(config["dataDir"]+"{sample}.fastq.
    gz")
```

and run snakemake.

Quality control, fastqc

```
if [ ! -d FastQC ]; then
   mkdir FastQC
fi
fastqc --outdir FastQC ${sample}.fastq.gz > FastQC/${sample
    }.log 2>&1
```

No more need to test the existence of a directory, it is created as needed.

rule fastqc:

This rule is already present in the snakefile

Reads mapping, bowtie2

```
if [ ! -d Bwt2_index ]; then
   mkdir Bwt2_index
   bowtie2-build ${genome} Bwt2_index/tauri > Bwt2_index/
    Bwt2_index.log 2>&1
fi
bowtie2 -x Bwt2_index/tauri -U ${sample}.fastq.gz -S ${
    sample}.sam > ${sample}_bowtie2.log 2>&1
```

2 rules: `genome_bwt2_index` (cf. previous ex.) and `bwt2_mapping`

ex2_o3.smk, rule bwt2_mapping (no run):

```
  output: "results/{sample}.sam"
  input: config["dataDir"]+"{sample}.fastq.gz",
         expand("Tmp/Otauri.{ext}.bt2", ext=BIDX)
  log: "Logs/{sample}_bwt2_mapping.log"
  shell: "bowtie2 -x Tmp/Otauri -U {input[0]} -S {output} 2>
     {log} "
```

Reads mapping, bowtie2

Why some troubles?

The snakemake launch probably didn't do what was expected. What have we forgotten?
We added a new rule to a snakemake but we didn't manage the rule tree, their is no input-output link to include the new rule to the workflow.
We will do that by completing the input directive of the target rule (caution to respect the Python "list" structure, coma-separated).

ex2_o2.smk, target rule:

```
rule all:
  input:
    expand("FastQC/{sample}_fastqc.html", sample=SAMPLES),
    expand("Tmp/Otauri.{ext}.bt2", ext=BIDX),
    expand("Tmp/{sample}.sam", sample=SAMPLES)
```

samtools

Shell script

```
samtools sort -O bam -o ${sample}_sort.bam ${sample}.sam
samtools index ${sample}_sort.bam
```

ex2_o4.smk, rule sam2bam_sort (no run):

```
  output:
    bam="Result/{sample}_sort.bam",
    bai="Result/{sample}_sort.bam.bai"
  input: "Tmp/{sample}.sam"
  log:
    sort="Logs/{sample}_sam2bam_sort.log",
    index="Logs/{sample}_bam2bai.log"
  shell:
    "samtools sort -O bam -o {output.bam} {input} 2> {log.
   sort} ;"
    "samtools index {output.bam} 2> {log.index}"
```

FeatureCount

Shell script

```
featureCounts -t gene -g ID -a ${annots} -s 1 -o ${sample}
    _ftc.txt ${sample}_sort.bam > ${sample}_ftc.log 2>&1
```

ex2_o5.smk, rule counting (params; no run):

```
  output: "Tmp/{sample}_ftc.txt"
  input:
    bam="Result/{sample}_sort.bam",
    annot=config["dataDir"]+config["annots"]
  params: t="gene", g="ID", s="1"
  log: "Logs/{sample}_counts.log"
  shell: "featureCounts -t {params.t} -g {params.g} -a {
    input.annot} -s {params.s} -o {output} {input.bam} &> {
    log}"
```

Counts matrix creation

Shell script

```
paste *_ftc.txt > counts_tmp.txt
awk -v nb=${nb_sample} 'BEGIN{FS="\t"}{count_tmp=$1; for(i
    =7;i<=nb*7;i=i+7){count=sprintf("%s\t%s",count_tmp,$i);
    count_tmp=count};print count}' counts_tmp.txt | sed 1d >
     counts.txt
```

Hint

Create 2 rules to manage some files aggregation to one result file:

- rule `extract_counts`: extract geneID and counts in individual files
- rule `matrix_counts`: paste these files

Counts matrix creation

ex2_o6.smk (2 rules, shell 3", copy, run):

```
rule matrix_counts:
  output: "Result/counts_matrix.txt"
  input: countfile=expand("Tmp/{sample}_ftc7.txt", sample=
    SAMPLES), geneID=expand("Tmp/{sample}_ftc1.txt", sample=
    SAMPLES)
  log: "Logs/matrix_counts.log"
  shell: """cp {input.geneID[0]} Tmp/ftc_geneID.txt > {log}
    ; paste Tmp/ftc_geneID.txt {input.countfile} > {output}
    > {log}"""

rule extract_counts:
  output: col7="Tmp/{sample}_ftc7.txt",
          col1="Tmp/{sample}_ftc1.txt"
  input: "Tmp/{sample}_ftc.txt"
  log: "Logs/{sample}_extract_counts.log"
  shell: """cut -f 7 {input} | sed 1d > {output.col7} > {log
    } ; cut -f 1 {input} | sed 1d > {output.col1} """
```

DESeq2

The DESeq2 step is the statistical analysis. From the count matrix, the statistical analysis is managed by a non parallelizable R script, DESeq2

Last challenge

Clean, delete and re-run !

```
cp ex2_o8.smk RNAseq_analysis.smk
cp ex2_o1.yml RNAseq_analysis_smkEnv.yml
rm -Rf FastQC/ Results/ Logs/ Tmp/
snakemake -s  RNAseq_analysis.smk --configfile
    RNAseq_analysis_smkEnv.yml
```

Bonus

Add a help rule

`https://lachlandeer.github.io/snakemake-econ-r-tutorial/self-documenting-help.html#a-help-rule`

Snakemake conclusion

Power gain

We have transposed the shell srcipt to a snakefile associated to a configuration file. This solution will be more powerful when we apply it in a High Performance Computing environment like the IFB cluster.

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