Medical Genomics Practical #1: Building reproducible workflows

N. Alcala & B. Alberti

Rare Cancers Genomics Team

November 8th & 15th 2023

International Agency for Research on Cancer

World Health Organization

RARE CANCERS GENOMICS

Plan

Part I. Practicals: generating a multi-omic Tumor Map of rare lung tumors

- **Concepts**: Lung Neuroendocrine Neoplasms
- **Data**

Part II. Introduction to domain-specific languages for bioinformatics

- **Concepts**: Reproducibility, scalability, portability
- **Exercises:** run a simple script

Part III. Create your first nextflow script

- **Concepts**: channels and processes
- **Exercises:** create a script printing "hello world"

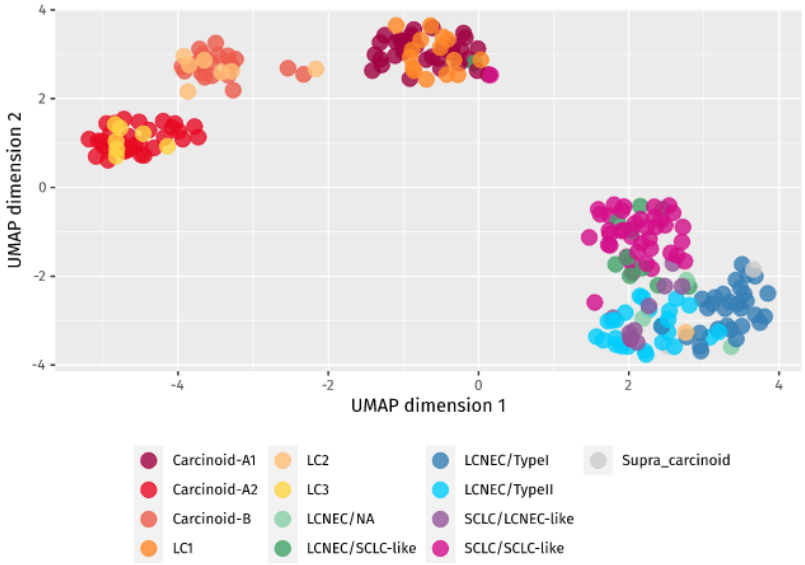
Part IV. Code a simple RNA-seq processing workflow

- **Concepts**: publish output, chaining processes, debugging
- **Exercises:** code a pipeline with multiple steps

Part I. Medical Genomics practicals | *General goal*

Generating a multi-omic Tumor Map of rare lung tumors

- **Lung Neuroendocrine Neoplasms** (LNENs) are rare solid cancers originating from pulmonary neuroendocrine cells
- They are classified (WHO) into grade 1 (typical carcinoids), grade 2 (atypical carcinoids), and grade 3 (carcinoma)
- **Multi-omic datasets** (exomes, RNA-seq, and methylation arrays) were recently generated
- **Clinically relevant molecular groups** with different prognosis (from 88% to 30% 10-year survival) and potential therapeutic targets



Tumor Map (UMAP) of lung neuroendocrine neoplasms. *Source:* https://nextjournal.com/rarecancersgenomics/a-molecular-map-of-lung-neuroendocrine-neoplasms/ *(Gabriel AAG*, Mathian E*, et al. GigaScience 2020). *Equally contributing.*

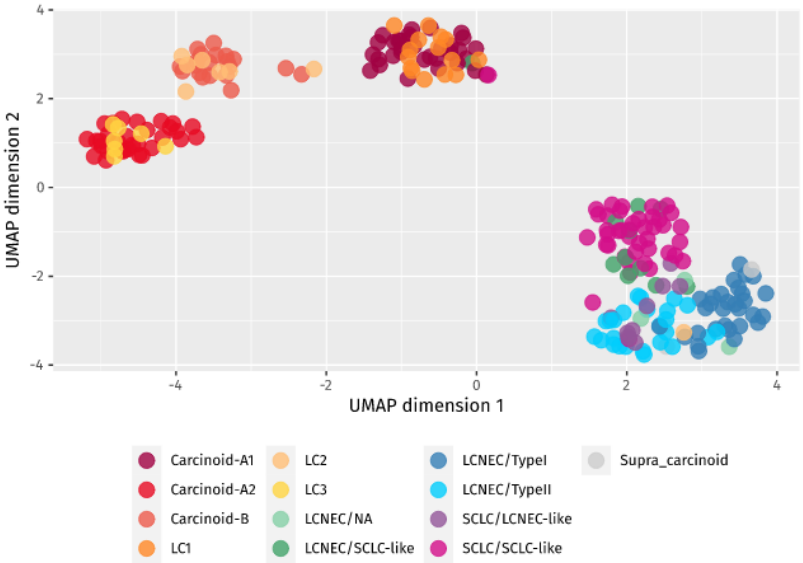
International Agency for Research on Cancer

World Health Organization

Part I. Medical Genomics practicals | *General goal*

Generating a multi-omic Tumor Map of rare lung tumors

- **Weeks 1-2: build a workflow to pre-process the RNA-seq data and obtain a gene expression matrix**
- **Week 3-4: generate a "tumor map" from the gene expression and gene methylation matrices, analyze the results**



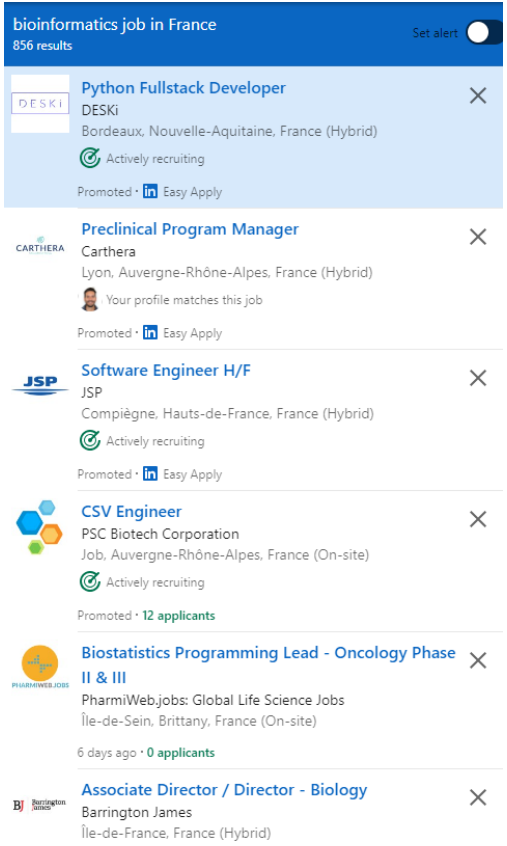
Tumor Map (UMAP) of lung neuroendocrine neoplasms. *Source:* https://nextjournal.com/rarecancersgenomics/a-molecular-map-of-lung-neuroendocrine-neoplasms/ *(Gabriel AAG*, Mathian E*, et al. GigaScience 2020). *Equally contributing.*

International Agency for Research on Cancer

World Health Organization

Part II. Introduction to DSL for bioinformatics | *Careers*

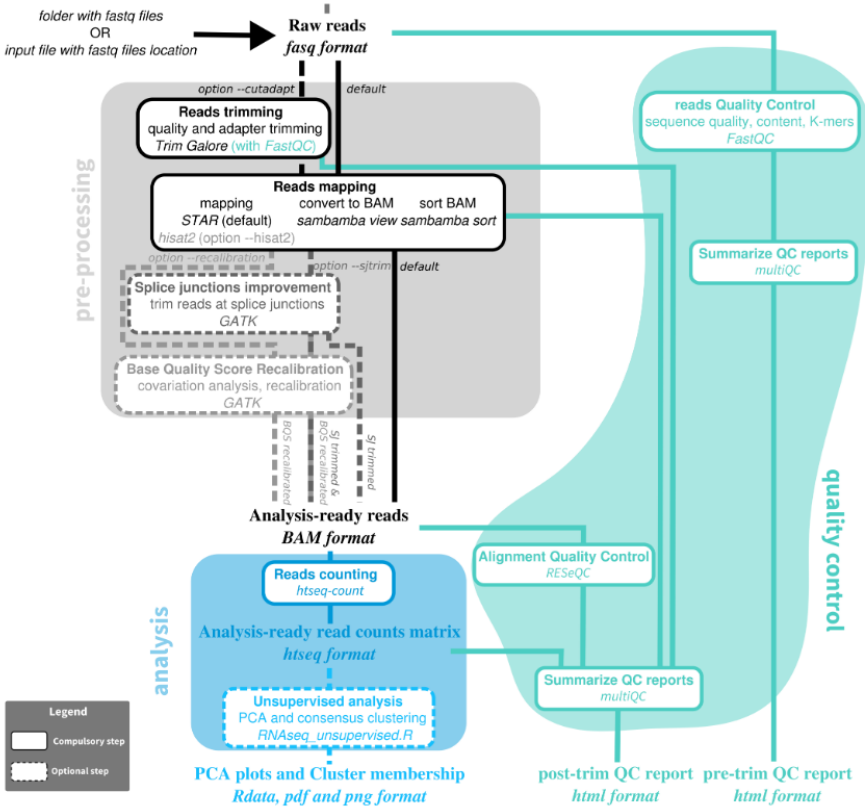
- Computational biology/bioinformatics careers range from **software engineering** to **data management/processing** and **data science**



Part II. Introduction to DSL for bioinformatics | *Example*

RNA-sequencing processing (IARCbioinfo/RNAseq-nf)

- Multiple tasks have to be performed in succession (e.g., trimming, mapping, quantification) or in parallel (on each sample)
- Each task uses its own software and libraries, and can have incompatible requirements (e.g., python versions)



International Agency for Research on Cancer

World Health Organization

Part II. Introduction to DSL for bioinformatics | *Concepts*

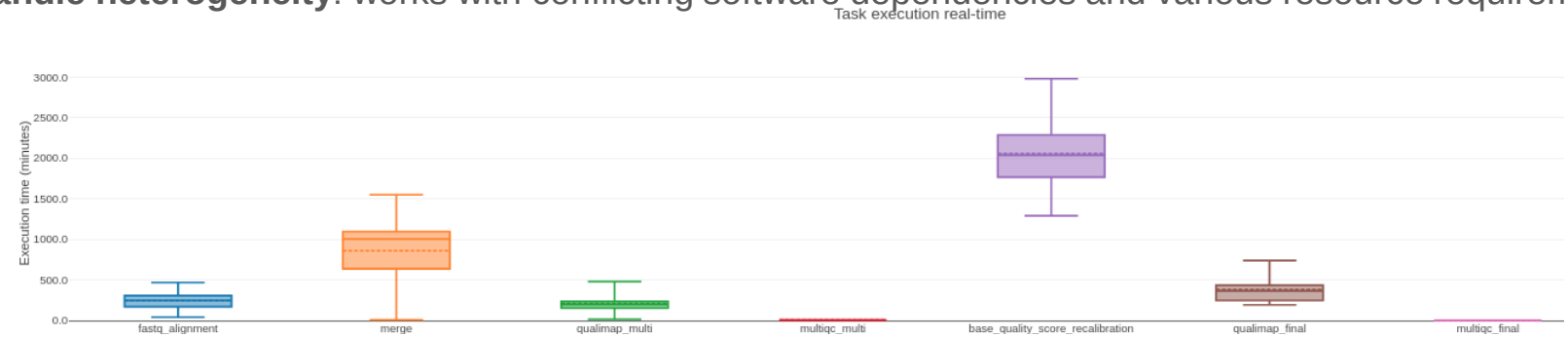
Workflow requirements for large-scale medical genomic projects

Reproducible: clinical and research applications need to be entirely reproducible

Scalable: easily run on large High-Performance Computing facilities

Portable: can run on various infrastructures (different OS, cloud)

Handle heterogeneity: works with conflicting software dependencies and various resource requirements



Example of duration of pre-processing. Tumor/normal pairs whole-genome sequencing (30X and 90X, respectively) for 10 patients, processed using workflow IARCbioinfo/alignment-nf (bwa+post-alignment+GATK BQSR+ pre- and post-alignment QC). **Total of 28,862.4 CPU hours (3 years and 3.5 months), actually processed in 3 days and 18 hrs.**

Part II. Introduction to DSL for bioinformatics | *Concepts*

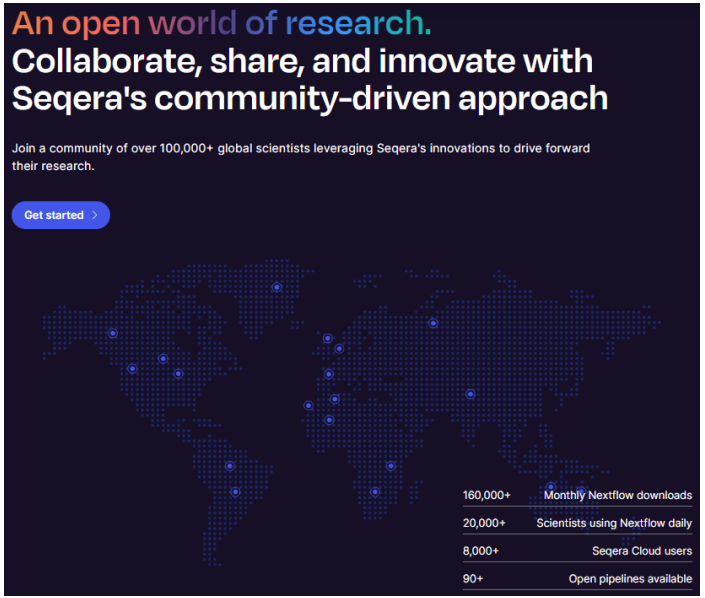
Solutions: domain-specific languages for bioinformatics

High-level programming languages designed to build workflows

- **Nextflow**: based on groovy (java); made at Centre for Genomic Regulation (Barcelona) / Seqera labs
- Snakemake: based on python; made at Institute of Human Genetics (Essen)
- WDL: own language; made at BROAD institute (Boston)

Part II. Introduction to DSL for bioinformatics | *Concepts*

Seqera labs (https://seqera.io/) Nextflow



International Agency for Research on Cancer

World Health Organization

Part II. Introduction to DSL for bioinformatics | *Concepts*

Nextflow design

1. Write code in any language

```
trim_galore --paired --fastqc --gzip \
  --basename sample1 -j 2 pair_1.fq pair_2.fq
```

2. Define succession of processes using dataflow programming

`Trimming | Mapping | Quantification`

3. Define software dependencies

`Conda, Docker, Singularity`

4. Define configuration

`Executor (local, HPC scheduler), error management, etc`

5. Run

Part II. Introduction to DSL for bioinformatics | *Concepts*

Nextflow language features

Dataflow language with implicit parallelization and scheduling of tasks:

- the **same set of tasks is applied to all input files**
- tasks are **automatically executed in the proper order** given their inputs and outputs
- **automatically handles submission** of jobs to HPC scheduler

Part II. Introduction to DSL for bioinformatics | *Concepts*

Running a workflow

```
nextflow run IARCbioinfo/pipeline-nf -profile conda --input_folder input/
```

Workflow name: either the *path to a nextflow script*, a *path to a directory* containing a nextflow config file pointing to a nextflow script, or a *github repository* (here, https://github.com/IARCbioinfo/pipeline-nf)

Nextflow options ("-"): e.g., profiles defining location of conda recipes listing software dependencies

Workflow options ("--"): e.g., input files location

Part II. Introduction to DSL for bioinformatics | *Practice*

Practical 1. Questions 1-3

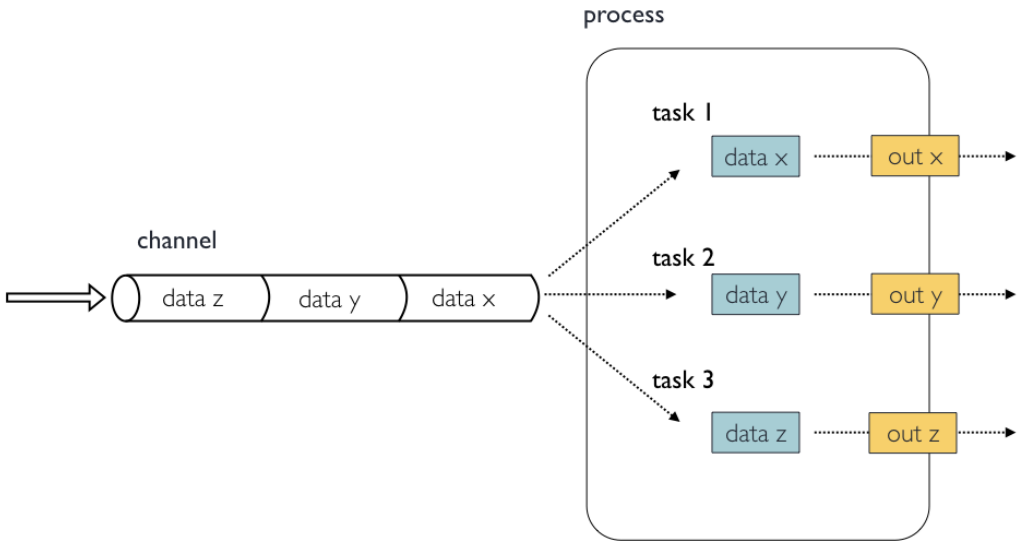
https://github.com/IARCbioinfo/medical_genomics_course/wiki/Practical-1

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Channel:** array storing process inputs or outputs (e.g., paths to data)

- **Processes:** instructions to execute a task on an input channel element
  - Note: Nextflow creates a work directory for each task, creates symbolic links to paths from Channel and executes the script defined in the process



Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Channel:** array storing process inputs or outputs (e.g., paths to data)

Example: test.nf containing

```
TOTO = Channel.from("toto") //creates channel
              .view()       //prints channel
```

```
> nextflow run test.nf
N E X T F L O W  ~  version 20.10.0

Launching `test.nf` [loving_roentgen] -
revision: 9634ca93db

toto
```

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Processes:** instructions to execute a task on an input channel element
  - name
  - input channel(s) name(s) and type(s)
  - output channel(s) name(s) and type(s)
  - script (code to be executed)

```
process test{
    input:
    path bam

    output:
    path "${bam}.name"

    shell:
    '''
    echo "!{bam}" > !{bam}.name
    '''
}
```

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Processes:** instructions to execute a task on an input channel element
  - input channel(s) type(s) and name(s)
  - output channel(s) type(s) and name(s)
  - types: *path (absolute path of files), val (for variables, e.g., strings & numbers), tuple (multiple values of different types), stdout (everything printed by the script)*
  - *supports wildcards*
  - *"$" to use nextflow variables*
  - *output paths are relative to the work folder*

```
process test{
        input:
        path bam

        output:
        path "${bam}.name"

        shell:
        '''
        echo "!{bam}" > !{bam}.name
        '''
}
```

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Processes:** instructions to execute a task on an input channel element
  - script (code to be executed)
  - “!” is used for nextflow variables (e.g., input variables) and “$” for bash variables

```
process test{
        input:
        path bam

        output:
        path "${bam}.name"

        shell:
        '''
        echo "!{bam}" > !{bam}.name
        '''
}
```

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **workflows:** order of processes to execute (similar to "main" in C language)

```
bam_ch = channel.fromFile("file1.bam")

process test{
...
}

workflow {
 test(bam_ch)
}
```

Part III. Coding a simple workflow | *Basics of Nextflow*

Coding a workflow

- **Nextflow scripting language:**
  - groovy syntax + nextflow-specific objects and operators
  - see more at https://www.nextflow.io/docs/latest/script.html#script-page

Part III. Coding a simple workflow | *Write a simple script*

Practical 1. Question 4

https://github.com/IARCbioinfo/medical_genomics_course/wiki/Practical-1

Part III. Coding a simple workflow | *Write a simple script*

Practical 1. Question 4

- **Correction**

```
params.greeting = "Hello World!"
greetch = channel.from(params.greeting)

process sayhello(){
        input:
                val x
        output:
                stdout
        shell:
        '''
        echo !{x}
        '''
}
workflow {
   sayhello(greetch).view()
}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow

- **Channel:** array storing process inputs or outputs (e.g., paths to data)
- **Channel factory:**
  - create a channel from a file path (note: can contain wildcards): `Channel.fromPath('data_test/BAM/*.bam')`

```
> N E X T F L O W  ~  version 20.10.0
Launching `test.nf` [deadly_yalow] - revision: e4af11bd16
/home/nalcala/data_test/BAM/NA06984_N.bam
/home/nalcala/data_test/BAM/NA06984_T.bam
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow

- **Channel:** array storing process inputs or outputs (e.g., paths to data)
- **Channel factory:**
  - create a channel from file pairs:
  `Channel.fromFilePairs('data_test/BAM/*.{bam,bam.bai}')`
  - see complete list of channel constructors at https://www.nextflow.io/docs/latest/channel.html#channel-factory

```
> N E X T F L O W  ~  version 20.10.0
Launching `test.nf` [mighty_rubens] - revision: 8e42d3ebe8
[NA06984_N, [/home/nalcala/data_test/BAM/NA06984_N.bam, /home/nalcala/data_test/BAM/NA06984_N.bam.bai]]
[NA06984_T, [/home/nalcala/data_test/BAM/NA06984_T.bam, /home/nalcala/data_test/BAM/NA06984_T.bam.bai]]
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow

- **Channel:** array storing process inputs or outputs (e.g., paths to data)
- **Channel factory:**
  - see complete list of channel constructors at https://www.nextflow.io/docs/latest/channel.html#channel-factory
- **Notes:**
  - *queue* Channels are consumed each time they are used
  - *singleton* Channels are reusable; created with `file()` and some other operators

```
ref = file("reference.fasta")
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow

- **Processes:**
  - multiple inputs can either be different channels (usually, one queue and all others are singletons) OR a tuple (vector) with multiple matching values

```
process test{
    input:
    path bedfile
    tuple val(name), path(bam)

    ...

}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow

- **Processes:**
  - multiple inputs can either be different channels (usually, one queue and all others are singletons) OR a tuple (vector) with multiple matching values
  - additionally, input and output elements can themselves be vectors of similar types; elements can be accessed using [index]

```
process test{
        input:
        path bedfile
        path bambai

        ...
        shell:

        '''

        echo "!{bambai[0]} !{bambai[1]}"

        '''

}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow 2

- **Processes:**
  - (optional) path to output directory
  By default, makes a *symbolic link* to the path of the actual output; can change behavior by specifying a “mode” (e.g., copy, move)
  `publishDir "output", mode: ‘move’`
  Note: once moved, the data is inaccessible so needs to be done at the very end of the script

```
process test{
        input:
        path bam

        output:
        path "${bam}.name"

        publishDir "output"
        shell:
        '''
        echo "!{bam}" > !{bam}.name
        '''
}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow 2

- **Processes:**
  - (optional) path to output directory
  By default, makes a *symbolic link* to the path of the actual output; can change behavior by specifying a "mode" (e.g., copy, move)
  `publishDir "output", mode: 'move'`
  *Note: **other directives** include the memory usage or number of cpus to use (see list at* https://www.nextflow.io/docs/latest/process.html#directives*)*

```
process test{
        input:
        path bam

        output:
        path "${bam}.name"

        publishDir "output"
        shell:
        '''
        echo "!{bam}" > !{bam}.name
        '''
}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow 3

- **workflows:** order of processes to execute (similar to "main" in C language), can use "piping" (|) syntax to chain processes

```
Nextflow.enable.dsl=2

bam_ch = channel.fromFile("file1.bam")

process preprocess{
...
}

process quantify{
...
}

workflow {
 preprocess(bam_ch) | quantify
}
```

International Agency for Research on Cancer

World Health Organization

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Coding a workflow 3

- **workflows:** order of processes to execute (similar to "main" in C language), can use "piping" (|) syntax to chain processes OR .out to get the output

```
Nextflow.enable.dsl=2

bam_ch = channel.fromFile("file1.bam")

process preprocess{
...
}

process quantify{
...
}

workflow {
 preprocess(bam_ch)
 quantify(preprocess.out)
}
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Passing parameters to nextflow script

- **Any variable VAR defined as "params.VAR" can be set at execution using flags "--var"**
- Its default value is set at declaration

Example script:

```
params.input_folder = "default_dir/"
Inputs = Channel.fromPath(params.input_folder)
...

> Nextflow run test.nf --input_folder BAM/
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Debugging a nextflow script

- **If error in the nextflow code:** usually an error is returned right away, processes are not executed; to debug: print values and channels (remember to duplicate with operator ".into()" so the printed channel is not consumed)
- **If error in script:** visit work directory created by nextflow and check files
  - *.command.sh*: actual script run
  - *.command.log, out, err*: log, output and error of script

```
name: N E X T F L O W  ~  version 20.10.0
Launching `test.nf` [extravagant_heisenberg] - revision:
d3ac3534eb
executor >  local (2)
[b9/d63363] process > test (2) [100%] 2 of 2, failed: 2 ✗
Error executing process > 'test (1)'
Caused by:
  Missing output file(s) `NA06984_T.bam.name` expected by
process `test (1)`
Command executed:
  echo "!{bam}" > !{bam}.name
Command exit status:
  0
Command output:
  (empty)
Work dir:
/home/nalcala/Medical_Genomics_TP1/work/b1/9dc097b6e17e218
52f1e969075a92d
```

Part IV. Coding an RNA-seq workflow | *Basics of Nextflow*

Resuming an execution

- **The "-resume" flag restarts restarts the execution**
- It checks which tasks were completed (they are cached)
- Processes with **changed parameters** will be re-run
- Processes with **modified code** will be re-run
- *Note: to be able to resume execution, nextflow keeps all intermediate files in its work directory, which needs to be cleaned periodically*

```
> Nextflow run test.nf -resume
```

Part IV. Coding an RNA-seq workflow | *Write a workflow*

Practical 1. Questions 5-7 + bonus questions

https://github.com/IARCbioinfo/medical_genomics_course/wiki/Practical-1

Part V. Coding an RNA-seq workflow | *Basics of Nextflow*

Being a nice user!

- **Computing facilities are shared resources, everyone needs to be mindful**

```
> export NXF_WORK=/temp/nalcala
```

Best practices:

- Work directory should be in the temporary folders (not $HOME), e.g., scratch or temp folders
- Use nextflow environment variables to set the right paths
- Always specify “cpus” and “mem” directives, and adapt resource usage to the system

Part VI. Going further | *Advanced Nextflow*

Interaction with github

- github allows **open code sharing**, **versioning, automated tests**
- **Nextflow can use specific revisions from github, corresponding to branches, tags, or commit IDs**

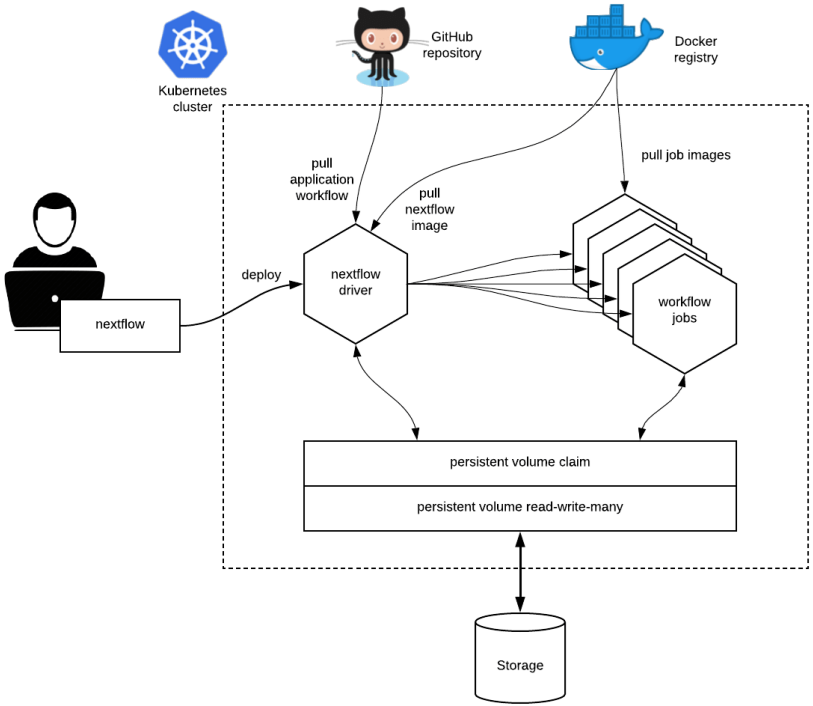
```
> Nextflow run IARCbioinfo/FastQC-nf -r v1.1 --help

> Nextflow run IARCbioinfo/FastQC-nf -r 06365b4dd152d586d2fb9e128c0188a5f84fd257 --help
```

Part VI. Going further | *Advanced Nextflow*

Use on other facilities

- Simply **setting the executor to the HPC scheduler** (e.g., lsf, slurm) enables using the HPC
- Can also be configured to use cloud resources (amazon web services, kubernetes, ...)



Part VI. Going further | *Advanced Nextflow*

- Possibility of using **modules** (external scripts containing some functions)
- Nextflow tower.nf provides an interface to **monitor job execution** and now also **launch jobs** (similar to Galaxy)

```
nextflow.enable.dsl=2

include { foo } from './some/module1'
include { bar } from './some/module2'

workflow {
   channel.from('Hello') | map { it.reverse() } | (foo &
bar) | mix | view
}
```

Part VI. Going further | *Advanced Nextflow*

Configuring nextflow

- **nextflow.config files:** contains options
  - nextflow concatenates config file given using the “-c” flag with the *nextflow.config* from current directory, that from the workflow directory and the global config at $HOME/.nextflow/config
  - useful to define *profiles* with options to run on different infrastructures
  - multiple profiles can be provided using the “-p” flag (e.g., “nextflow run -p conda -p SLURM_IARC”)

```
profiles {
  conda {
          process.conda =
"$baseDir/environment.yml"
  }
  Singularity {
          singularity.enabled = true
          process.container =
'shub://IARCbioinfo/fastqc-nf:v1.1'
  }
  SLURM_IARC {
          process.executor = “slurm”
          queue = “low_p”
          queueSize = 50
  }
}
```

Part VI. Going further | *Advanced Nextflow*

Configuring nextflow

- **nextflow.config files:** contains options for running nextflow.
- **Best practices for reproducibility:** create a conda recipe (yml file) containing all software dependencies

File environment.yml:

```
name: GATK4
channels:
  - bioconda
  - conda-forge
dependencies:
  - gatk4=4.1.5.0
```

Medical Genomics Practical #1: Building reproducible workflows

International Agency for Research on Cancer
Lyon, France

Appendix

International Agency for Research on Cancer

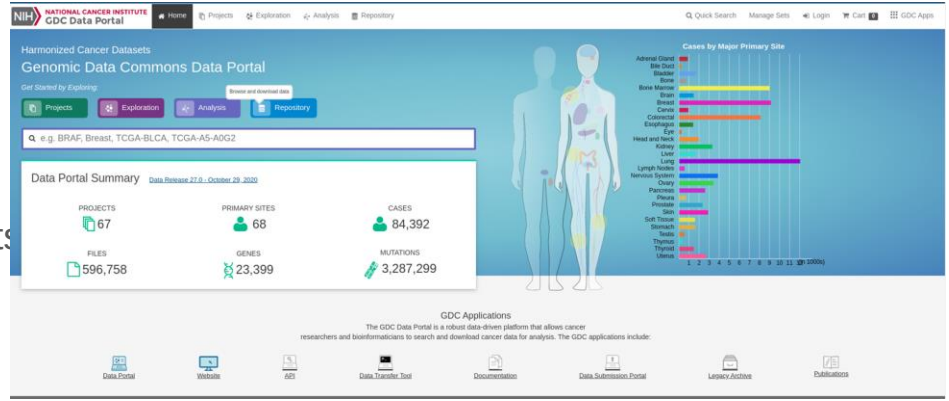
World Health Organization

Part I. Transcriptomics | *Ressources*

The Cancer Genome Atlas (TCGA) project

Database of cancer multi-omic data for

- Tumors from 33 primary sites
- RNA-seq data under controlled access (requires research institute affiliation)
- Processed gene expression data (read counts and FPKM) open-access



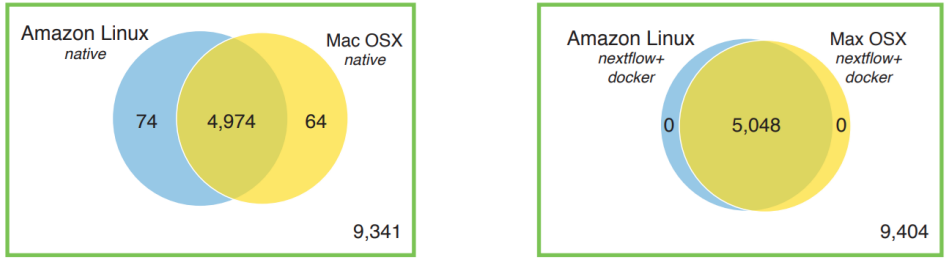
Web interface of the genomic data portal hosting the TCGA data. *Source:* https://portal.gdc.cancer.gov/.

Part I. Transcriptomics | *Techniques*

Bulk sequencing: processing

Notes:

- *RNA-seq analyses are known to suffer from a lack of robustness, so* ***reproducibility and open science practices are of the uttermost importance****!*
- *Even subtle numerical instability issues can impact the results (****Figure****)*
- *See* ***Practical 1*** *for a solution to provide entirely reproducible workflows*



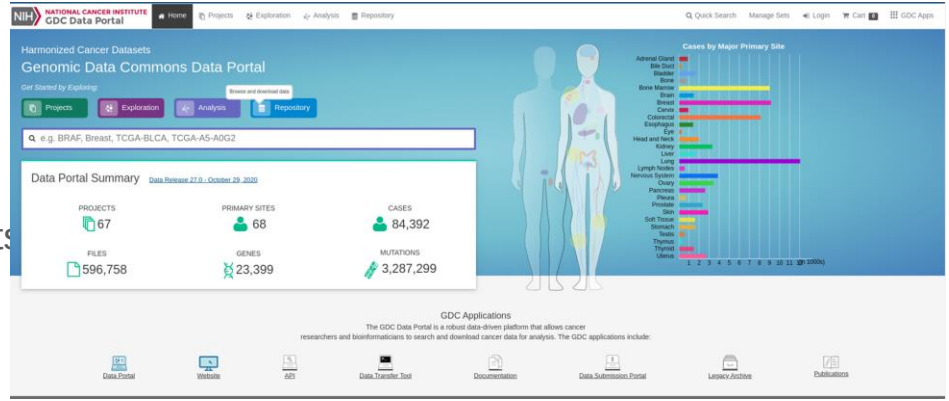
Reproducibility of RNA-seq differential expression analysis of Human Lung Fibroblasts. **Left.** Venn diagram of differentially expressed genes using the exact same versions of all softwares (kallisto and sleuth) but two different OS (blue: a Linux system from AWS, yellow: Mac OS). **Right.** Same as on the left, but running the entire data processing and analysis workflow inside a docker container using the Nextflow language. *Source: Di Tommaso et al. Nature Biotechnology 2017.*

Part I. Transcriptomics | *Ressources*

The Cancer Genome Atlas (TCGA) project

Database of cancer multi-omic data for

- Tumors from 33 primary sites
- RNA-seq data under controlled access (requires research institute affiliation)
- Processed gene expression data (read counts and FPKM) open-access



Web interface of the genomic data portal hosting the TCGA data. *Source:* https://portal.gdc.cancer.gov/.

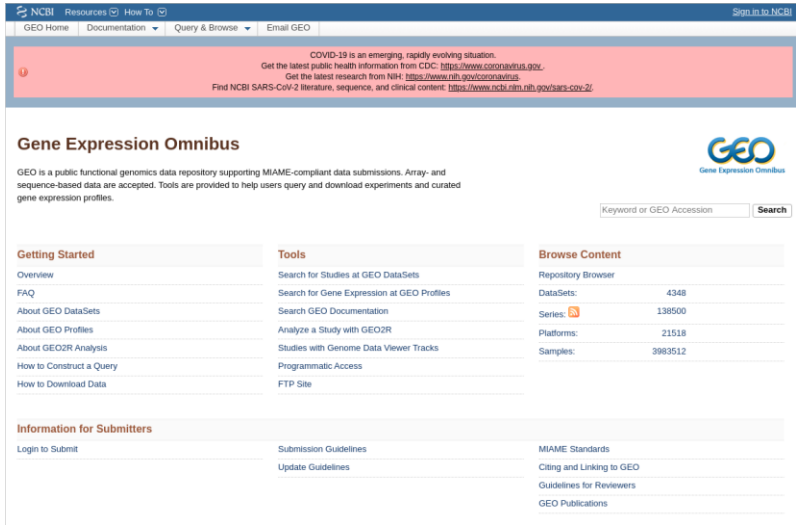
Part I. Transcriptomics | *Ressources*

The Gene Expression Omnibus (GEO) repository

Database of expression data (arrays and RNA-seq)

- Includes human data
- All data is open-access

Will be used for the practicals.



Web interface of the gene expression omnibus repository. *Source:* https://www.ncbi.nlm.nih.gov/geo/.