

# HPC-Pipes Nextflow

from the  
Health Data Science  
Sandbox



**Alba Refoyo Martinez, PhD**

Data scientist

Center for Health Data Science (HeaDS)

UNIVERSITY OF  
COPENHAGEN





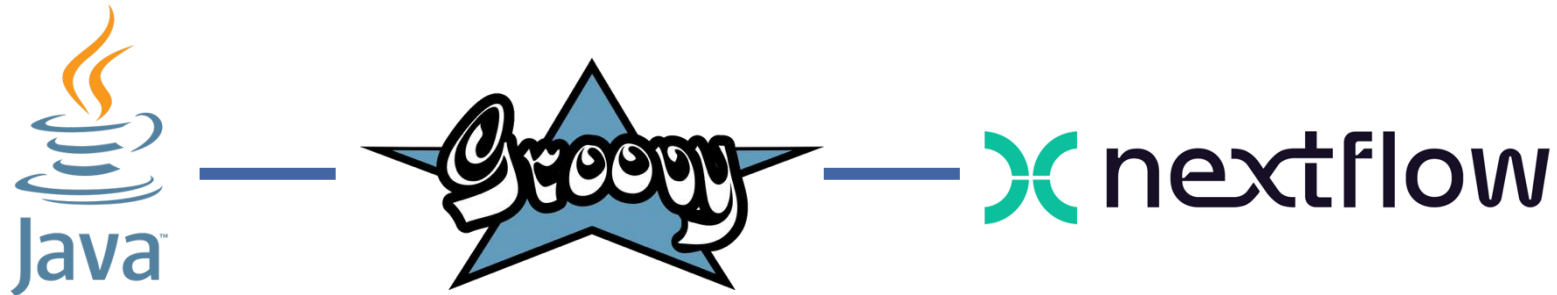
## I - Basics

Understand elements  
of a nextflow pipeline

Understand how to run  
and resume nextflow  
pipeline

# What is Nextflow?

A tool for managing scientific workflows, written in groovy, a language for java program



A **dataflow** programming model

- Communication by dataflow variables
- Processes (software/scripts) receiving (inputs) and emitting (outputs) through channels



## Why Nextflow?

Lively & growing user community (nf-core on Slack)

Large library of ready-made pipelines

Easy to use (ready made pipelines)

Support by industry (Sequera, Elixir)

Developers “kinda” like it (templates available)

...

Another way to fight unemployment (nf developers needed)



## Core features

- A workflow manager
- Reproducibility (integration with containers and GitHub)
- Portability (schedulers, cloud platforms...)
- Parallelisation
- Scalability
- Easy to **resume** (monitors each chunk/file and process)
- Easy prototyping (add extra steps, reuse your existing scripts and tools)



# Resources

Documentation: <https://www.nextflow.io/docs/latest/>

Training material: [https://training.nextflow.io/latest/hello\\_nextflow/](https://training.nextflow.io/latest/hello_nextflow/)



## Nextflow BioHelper

By Mr robert king 8

Assists in creating Nextflow DSL2 pipelines for bioinformatics

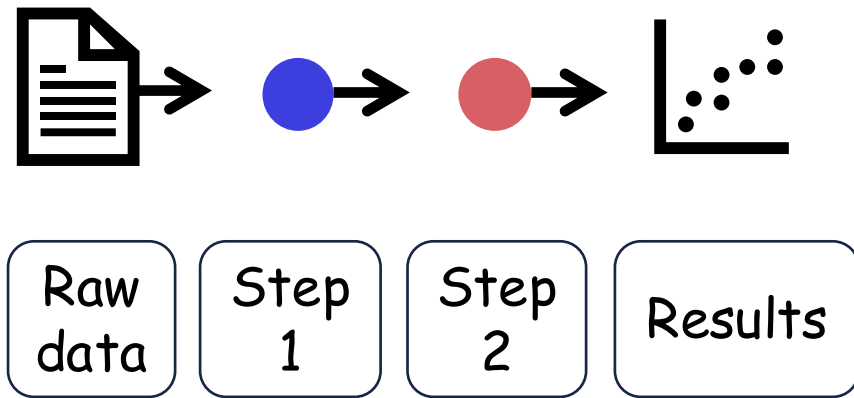
<https://chatgpt.com/g/g-gNHgmDFKu-nextflow-biohelper>



# Workflows Processes Channels Operators Config File

Nextflow building blocks: process and channel

The workflow is defined in terms of **processes** that define how to create output **channels** from input **channels**



main.nf

```
workflow {  
    first_process()  
    second_process()  
}
```



# Workflows Processes Channels Operators Config File

## A process: structure

```
process < name > {  
  [ directives ]  
  
  input:  
  < process inputs >      / <input channel> /  
  
  output:  
  < process outputs >      / <output channel> /  
  
  when:  
  < condition >  
  
  [script|shell|exec]:      / <task> /  
  ""  
  # scripts/commands to run  
  ""  
}
```

A **process** is the basic processing primitive to execute a user script.

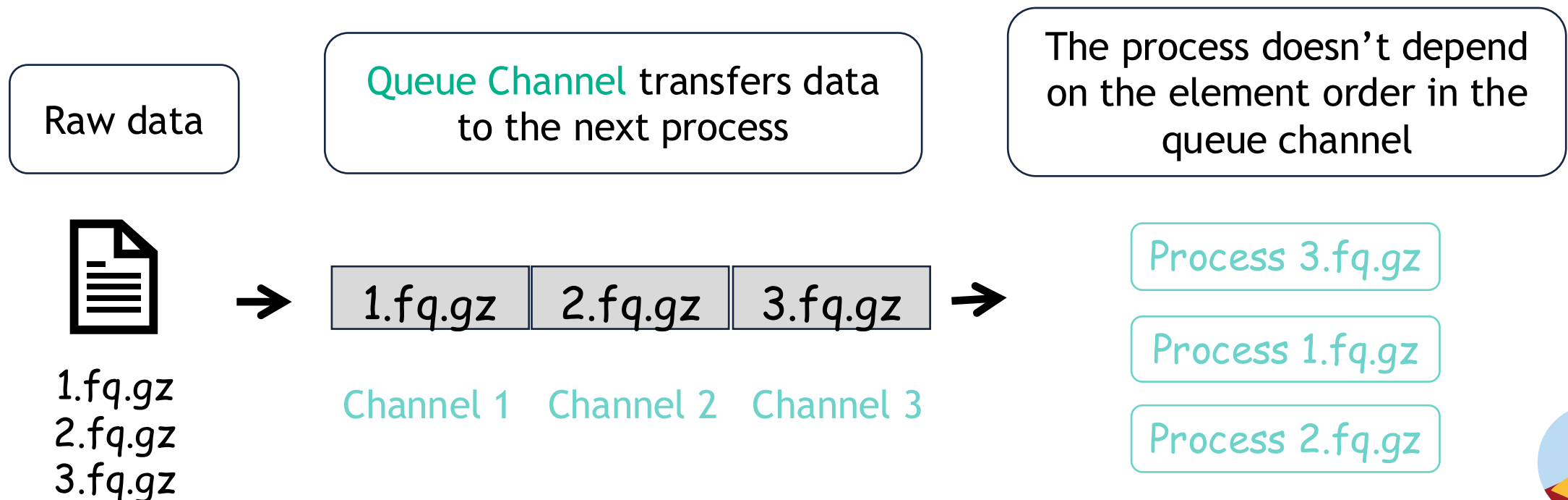
Each process is one (independent) step in the pipeline block.



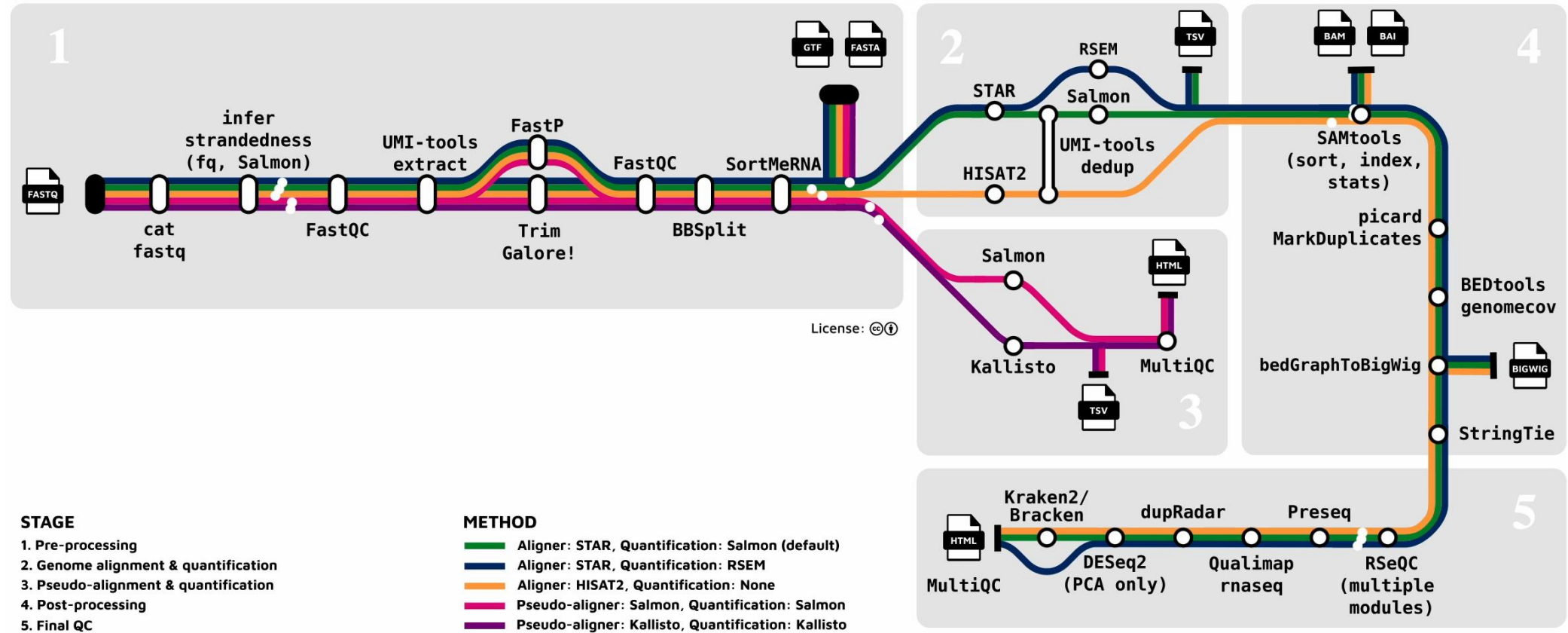


**Information flows** from one process to another via ‘**channels**’ as defined in the input and output sections of each process.

Tasks will automatically trigger when all required data inputs are available from the channels.



# Workflows Processes Channels Operators Config File



[Source to animation: nf-core/rnaseq/](https://nf-core.org/rnaseq/)



Workflows

Processes

**Channels**

Operators

Config File

**Value channel:** can be used multiple times in workflow (shared across processes).

- **Channel.value** (e.g. single value, list object, map object)

**Queue channel:** consumed when they are used by a process or operator

- **Channel.of(x)** **/single values/**
- **Channel.empty()**
- **Channel.fromList(['a','b'])** **/lists/**
- **Channel.fromPath("data/\*.txt")** **/path2Files/**
- **Channel.fromPath("data/\*\_{1,2}.fastq.gz").filePairs()** **/paired-end/**

```
channel.<method>
```



# Channel operations: creating input channels from list

## Example 1

`Channel.of( "A.fa", "B.fa", "C.fa" )`

A.fa →  
B.fa →  
C.fa →

- **Each item (row) has 1 execution of the process**
- Here is 1 channel with 3 items
- 3 parallel executions of the process

`Channel.of( [ "A.fa", "B.fa", "C.fa" ] )`

[ A.fa, B.fa, C.fa ] →

- 1 channel, 1 item, 1 execution

## Example 2

input [ A.fa, B.fa, C.fa ]

`input.flatten()`

A.fa →  
B.fa →  
C.fa →

- 1 channel, 3 parallel executions of the process
- Note difference vs `fromFilePairs( ... flat:true )`

input

A.fa  
B.fa  
C.fa

`input.collect()`

`input.toSortedList()`

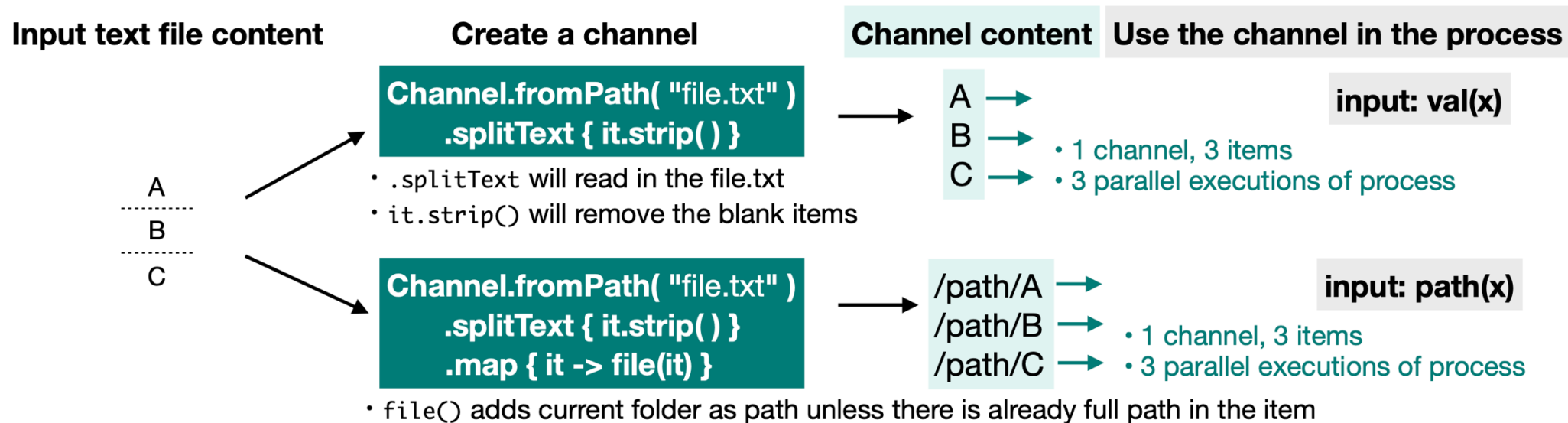
[ A.fa, B.fa, C.fa ]

- `.collect()` will maintain the order of items in the input channel, but if that channel is generated by a process, item order may be unpredictable
- `.toSortedList()` will sort the items

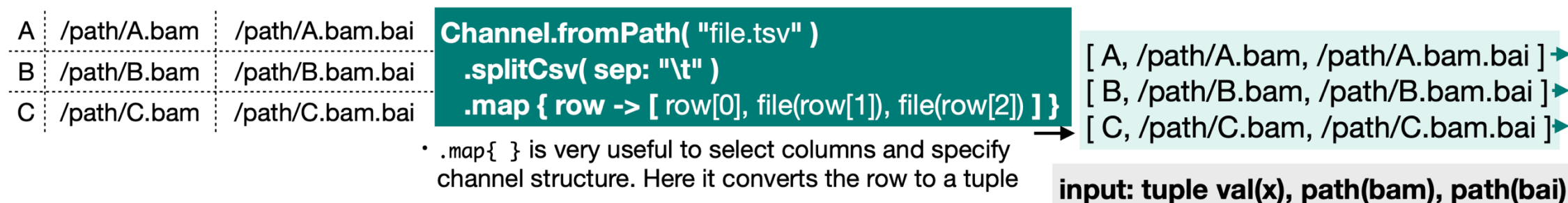


# Channel operations: creating input channels from text files

## Example 1

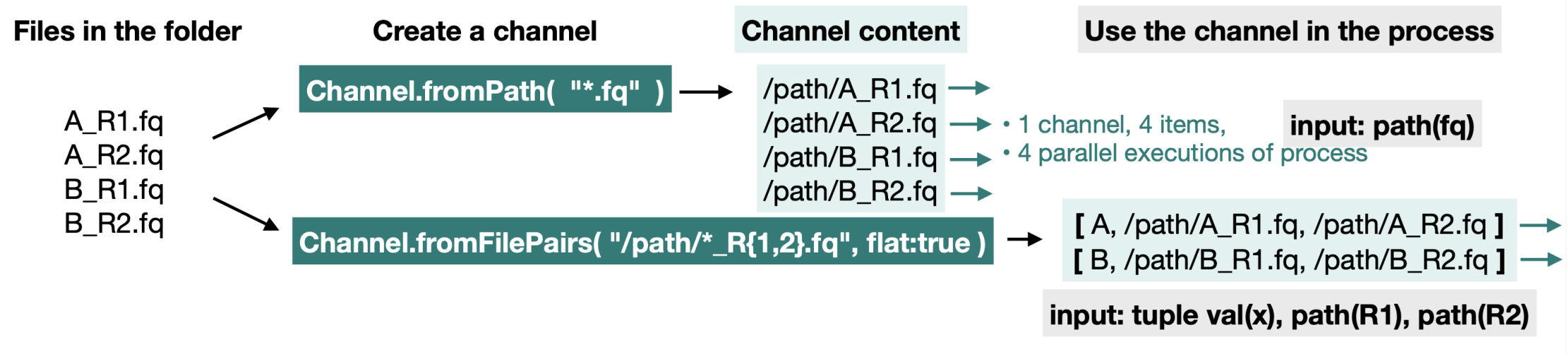


## Example 2



# Channel operations: creating input channels from list of files

## Example 1



Other useful functions:

- `.collect()` and `.combine()`
- `.toList()` and `.concat()`



# Workflow example: Hello World

main.nf

```
#!/usr/bin/env nextflow
```

```
params.greeting = 'Hello world!'
greeting_ch = Channel.of(params.greeting)
```

```
process SPLITLETTERS {
    input:
    val x

    output:
    path 'chunk_*'

    script:
    """
    printf '$x' | split -b 6 - chunk_
    """
}
```

process CONVERTTOUPPER {  
input:  
path y

output:  
stdout

script:  
"""  
cat \$y | tr '[a-z]' '[A-Z]'  
"""

}

workflow {  
 letters\_ch = SPLITLETTERS(greeting\_ch)  
 results\_ch = CONVERTTOUPPER(letters\_ch.flatten())  
 results\_ch.view { it }  
}

Process name



# Nextflow usage



> nextflow [options] COMMAND [arg...]

## Options:

- C** Use the specified configuration file(s) overriding any defaults
- D** Set JVM properties
- bg** Execute nextflow in background
- c, -config** Add the specified file to configuration set
- config-ignore-includes** Disable the parsing of config includes
- h** Print this help
- log** Set nextflow log file path
- q, -quiet** Do not print information messages
- remote-debug** Enable JVM interactive remote debugging (experimental)
- syslog** Send logs to syslog server (eg. localhost:514)
- trace** Enable trace level logging for the specified package name - multiple packages can be provided separating them with a comma e.g. '-trace nextflow,io.sequera'
- v, -version** Print the program version

## Commands:

- clean** Clean up project cache and work directories
- clone** Clone a project into a folder
- config** Print a project configuration
- console** Launch Nextflow interactive console
- drop** Delete the local copy of a project
- help** Print the usage help for a command
- info** Print project and system runtime information
- inspect** Inspect process settings in a pipeline project
- kuberun** Execute a workflow in a Kubernetes cluster (experimental)
- list** List all downloaded projects
- log** Print executions log and runtime info
- plugin** Execute plugin-specific commands
- pull** Download or update a project
- run** **Execute a pipeline project**
- secrets** Manage pipeline secrets
- self-update** Update nextflow runtime to the latest available version
- view** View project script file(s)





# Nextflow usage



```
> nextflow run main.nf
```

## Output

```
N E X T F L O W ~ version 23.10.1
```

```
Launching main.nf [cheeky_keller] DSL2 - revision: 197a0e289a
```

```
executor > local (3)
```

```
[31/52c31e] process > SPLITLETTERS (1) [100%] 1 of 1 ✓
```

```
[37/b9332f] process > CONVERTTOUPPER (2) [100%] 2 of 2 ✓
```

```
HELLO
```

```
WORLD!
```



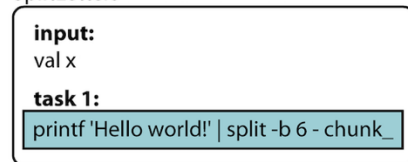
nextflow run hello.nf

```
> _input  
params.greeting = 'Hello world!'
```

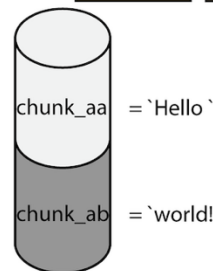
channel 1 :  
greeting\_ch



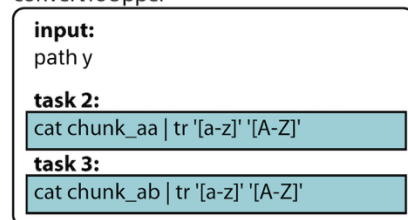
process 1:  
SplitLetters



channel 2 :  
letters\_ch  
.flatten()



process 2:  
convertToUpper



channel 3 :  
result\_ch



```
> _output  
result_ch.view{ it } = HELLO  
                      WORLD!
```

```
#!/usr/bin/env nextflow
```

```
params.greeting = 'Hello world!'  
greeting_ch = Channel.of(params.greeting)
```

```
process SPLITLETTERS {
```

```
    input:  
    val x
```

```
    output:  
    path 'chunk_*'
```

```
    script:
```

```
    """
```

```
    printf '$x' | split -b 6 - chunk_  
    """
```

```
}
```

```
:
```

```
:
```

```
:
```

```
:
```

```
workflow {
```

```
    letters_ch = SPLITLETTERS(greeting_ch)
```

```
    results_ch = CONVERTTOUPPER(letters_ch.flatten())
```

```
    results_ch.view { it }
```

```
}
```



# Modify and resume

## main.nf

```
#!/usr/bin/env nextflow

params.greeting = 'Hello world!'
greeting_ch = Channel.of(params.greeting)

process SPLITLETTERS {
    input:
    val x

    output:
    path 'chunk_*'

    script:
    """
    printf '$x' | split -b 6 - chunk_
    """
}
```

```
process CONVERTTOUPPER {
    input:
    path y

    output:
    stdout

    script:
    """
    rev $y
    """
}

workflow {
    letters_ch = SPLITLETTERS(greeting_ch)
    results_ch = CONVERTTOUPPER(letters_ch.flatten())
    results_ch.view { it }
}
```



## Modify and resume



```
> nextflow run main.nf -resume
```

### Output

```
N E X T F L O W ~ version 23.10.1
```

```
Launching main.nf [zen_colden] DSL2 - revision: 0676c711e8
```

```
executor > local (2)
```

```
[31/52c31e] process > SPLITLETTERS (1) [100%] 1 of 1, cached: 1 ✓
```

```
[0f/8175a7] process > CONVERTTOUPPER (1) [100%] 2 of 2 ✓
```

```
!dlrow
```

```
olleH
```

The task ID is the same as in the first output  
(results-files are cached)



# Parameterisation

Enables you to change the input at runtime

Declare in Nextflow script by prepending the prefix `params.<x>` to a variable (e.g.,  
`params.greetings = "Hello world"`)

Pass a pipeline parameter to the script on commandline using the variable name (e.g.,  
`nextflow run main.nf -greetings "Hei"`)

Use a file (yaml/json) to pass many parameters using option `-params-file`

main.nf

```
#!/usr/bin/env nextflow

params.greeting = 'Hello world!'
greeting_ch = Channel.of(params.greeting)

process SPLITLETTERS {
    ...
}
```



# Reproducibility: running Nextflow from GitHub



```
(base) gsd818@SUN1029429 ~ % nextflow run hello
```

```
NEXTFLOW ~ version 24.10.5
```

```
Pulling nextflow-io/hello ...
```

```
downloaded from https://github.com/nextflow-io/hello.git
```

```
Launching `https://github.com/nextflow-io/hello` [evil_lampport] DSL2 - revision: afff16a9b4 [master]
```

```
executor > local (4)
```

```
[77/1e7d00] sayHello (1) [100%] 4 of 4 ✓
```

```
Ciao world!
```

```
Hola world!
```

```
Hello world!
```

```
Bonjour world!
```



Workflows

Processes

Channels

**Operators**

Config File

## USE: manipulate and transform channels

<https://www.nextflow.io/docs/latest/reference/operator.html>  
(full list)

- Branch
- Collect
- Combine
- Concat
- Count
- CountFasta
- countLines
- Filter
- ....

```
Channel.of( 1, 2, 3, 1, 1, 2, 2, 3 )  
    .unique()  
    .view()
```

```
Channel.of( 'a', 'b', 'bc', 3, 4.5 )  
    .filter( Number )  
    .view()
```

```
Channel.of(1, 2, 3, 40, 50)  
    .branch { v ->  
        small: v < 10  
        large: v > 10 }  
    .set { result }  
  
result.small.view { v -> "$v is small" }
```



Workflows

Processes

Channels

Operators

Config File

## USE: manipulate and transform channels

<https://www.nextflow.io/docs/latest/reference/operator.html>  
(full list)

- Branch
- Collect
- Combine
- Concat
- Count
- CountFasta
- countLines
- Filter
- ....

```
Channel.of( 1, 2, 3, 1, 1, 2, 2, 3 )  
    .unique()  
    .view()
```

```
1  
2  
3
```

```
Channel.of( 'a', 'b', 'bc', 3, 4.5 )  
    .filter( Number )  
    .view()
```

```
3  
4.5
```

```
Channel.of(1, 2, 3, 40, 50)  
    .branch { v ->  
        small: v < 10  
        large: v > 10 }  
    .set { result }
```

```
result.small.view { v -> "$v is small" }
```

```
1 is small  
2 is small  
3 is small
```





USE: managing HPC resources

Nextflow can load pipeline configs from:

- Home directory
- Workflow directory (if not current dir)
- Current dir
- Config file -c <config.file>

SYNTAX:

- Block
- Dot

```
custom.config
```

**// block syntax**

```
process {  
    executor = 'slurm'  
    memory = '4 GB'  
    cpus = 2  
}
```

**// dot syntax**

```
process.executor = 'slurm'  
process.memory = '4 GB'  
process.cpus = 2
```



## Scopes:

- Env
- Params
- Process
- Executor
- Profiles: set of config attributes activated during pipeline execution ( -profile xxx )



```
> nextflow run ... -profile cluster
```

`custom.config`

```
process.queue='small'  
process.memory='10G'
```

```
singularity{  
    enabled=true  
}
```

```
profiles {  
    standard {  
        process.executor= local'  
    }  
    cluster {  
        process.executor= 'slurm'  
    }  
}
```

Example of 2  
profiles defined



Workflows

Processes

Channels

Operators

Config File

```
custom.config
```

```
// create compute envs specific to profiles (example)
```

```
profiles {  
  compute_cluster_01 {  
    process.clusterOptions = '-w compute_cluster_01'  
    queue_size = 10  
  }  
  compute_cluster_02 {  
    process.clusterOptions = '-w compute_cluster_02'  
    queue_size = 100  
  }  
}
```



```
> nextflow run ... -profile compute_cluster_02
```



Workflows

Processes

Channels

Operators

Config File

```
custom.config
```

```
// can define resources for specific tasks
```

```
process {  
  executor = 'slurm'  
  memory = '4 GB'  
  cpus = 2  
  
  withLabel: 'foo' { cpus = 2 }  
  withLabel: '!foo' { cpus = 4 }  
  withName: '!align.*' { queue = 'long' }  
  
}
```

## Optimization

Ideal for tasks that benefit from different CPU resource allocations



<https://www.nextflow.io/docs/latest/config.html#config-page>

## Hierarchy of config files:

1. The config file `$HOME/.nextflow/config` (or `$NXF_HOME/.nextflow/config` when `NXF_HOME` is set).
2. The config file `nextflow.config` in the project directory
3. The config file `nextflow.config` in the launch directory
4. Config files specified using the `-c <config-files>` option

### Tip

You can alternatively use the `-C <config-file>` option to specify a fixed set of configuration files and ignore all other files.

Soft override `-c` (lowercase)  
Hard override with `-C` (uppercase)



## Example running pipelines in a queueing system

my\_pipeline/

- |— <slurm-batch>.sh # Submission batch script
- |— <workflow-script>.nf # Nextflow script (e.g. main.nf)
- |— <config-file>.config # Configuration file for pipeline parameters and environment settings (e.g. nextflow.config)
- |— data # Input data (e.g. input.txt)
- |— scripts

You might want to change the default settings



# Slurm-batch.sh (part I)

```
#!/bin/bash
```

```
### slurm job options
```

```
#SBATCH --job-name=cactus_test # job name
```

```
#SBATCH --mail-type=END,FAIL # mail events (NONE, BEGIN, END, FAIL, ALL)
```

```
#SBATCH --mail-user=gsd818@ku.dk # email address to receive the notification
```

```
#SBATCH -c 1 # number of requested cores
```

```
#SBATCH --mem=2gb # total requested RAM
```

```
#SBATCH --time=1-00:00:00 # max. running time of the job, format in D-HH:MM:SS
```

```
#SBATCH --output=cactus_test_%j.log # standard output and error log, '%j' gives the job ID
```

```
#SBATCH -w dancmpn02fl
```

```
.  
.   
.
```



# Slurm-batch.sh (part II)

```
# load required modules for pipeline
module load python/3.8.16 java/11.0.15 singularity/3.8.0 nextflow/23.10.1

# setup bash env variables
export NXF_OPTS='-Xms1g -Xmx4g'
export NXF_VER=22.10.8
echo <NXF_HOME> $NXF_HOME           # preferably somewhere in ~/cache/nxf-home
echo <NXF_TEMP> $NXF_TEMP           # preferably somewhere in /scratch
echo <NXF_SINGULARITY_CACHEDIR> $NXF_SINGULARITY_CACHEDIR # ~/cache/singularity-images

# run cactus pipeline
nextflow run jsalignon/cactus \
  -r main -latest \
  -params-file parameters/params.yml \  # Define most params here
  -c /projects//people/gsd818/config-file.config \ # Define configuration/requirements
  -profile singularity \
  --disable_all_enrichments \           # Or other specific parameters from the pipeline
  -work-dir ./work \
  -resume
```





# config-file.config

```
params {  
  references_dir="/projects/data/references"  
}
```

```
singularity {  
  enabled=true  
  autoMounts=true  
  runOptions='--bind /projects:/projects'  
}
```

```
process {  
  errorStrategy='retry' # ignore  
  maxRetries=1  
  memory=72.GB  
  cpus= 12  
  time= 18.h  
  executor='slurm'  
}
```

```
executor {  
  queueSize= 7  
  submitRateLimit = '30/1min'  
}
```

```
profiles {  
  nodecmpn01fl {  
    process.clusterOptions = '-w nodecmpn01fl'  
  }  
  nodegpu01fl {  
    process.clusterOptions = '-p gpuqueue'  
  }  
}
```



## Example config published file

### NNF reNEW/GPR Genomics Platform

- HPC cluster: danhead
  - **Slurm** for resource management
  - **Singularity/Apptainer** for environment management
  - Profile: [https://nf-co.re/configs/ku\\_sund\\_danhead](https://nf-co.re/configs/ku_sund_danhead)



## Inspecting Nextflow results

Nextflow creates a folder (i.e., inside work directory for each process-  
using some hash numbers)

Each folder contains:

- Links to input files
- Output files
- Number of hidden files
- Script used for the process

You can publish results to a different folder



# Tips: running Nextflow pipelines

## Commandline

Distinguish between single hyphen `-` and double hyphen `--`

Difference between `-c` and `-C` and config hierarchy

ALWAYS add revision version `-r` for reproducibility

Use optional flags for creating reports and visualizations (`-with-dag`, `-with-timeline`, `-with-report`)

## Output

Clean up your temporary directories

Output for each task/module is stored in the working directory

Find work directory location with `$NXF_WORK`



# Tips: running Nextflow pipelines

## Clean up old temporary files

```
1 srun -c 2 -w dancmpn01fl --mem=8gb --time=0-00:30:00 --pty bash
2
3 # see how many jobs are older than 30 days
4 find $NXF_WORK/* -type d -ctime +30 | wc -l
5
6 # delete these jobs
7 find $NXF_WORK/* -type d -ctime +30 | xargs rm -rf
```

## Explore logs and troubleshoot jobs

```
1 # find failed runs
2 grep FAIL <outdir>/pipeline_info/execution_trace_*.txt
```

```
1 # start the interactive session on the node where computations took place
2 srun -c 2 --nodelist=dancmpn01fl --mem=8gb --time=0-00:05:00 --pty bash
3 JOBID="e7/bb97b7"
4 less $NXF_WORK/$JOBID*/.command.err
```



# Comparing Workflow Managers

## Snakemake

Language	Python
Data	Everything is a file
Execution	Working directory
Philosophy	“Pull” - desired outputs <code>snakemake results/plot.png</code>
Dry runs	Yes (snakemake -n)
Track code changes	Timestamps and hashes rule logic (tracked in .snakemake)
Data constructions	Reverse (output -> input)

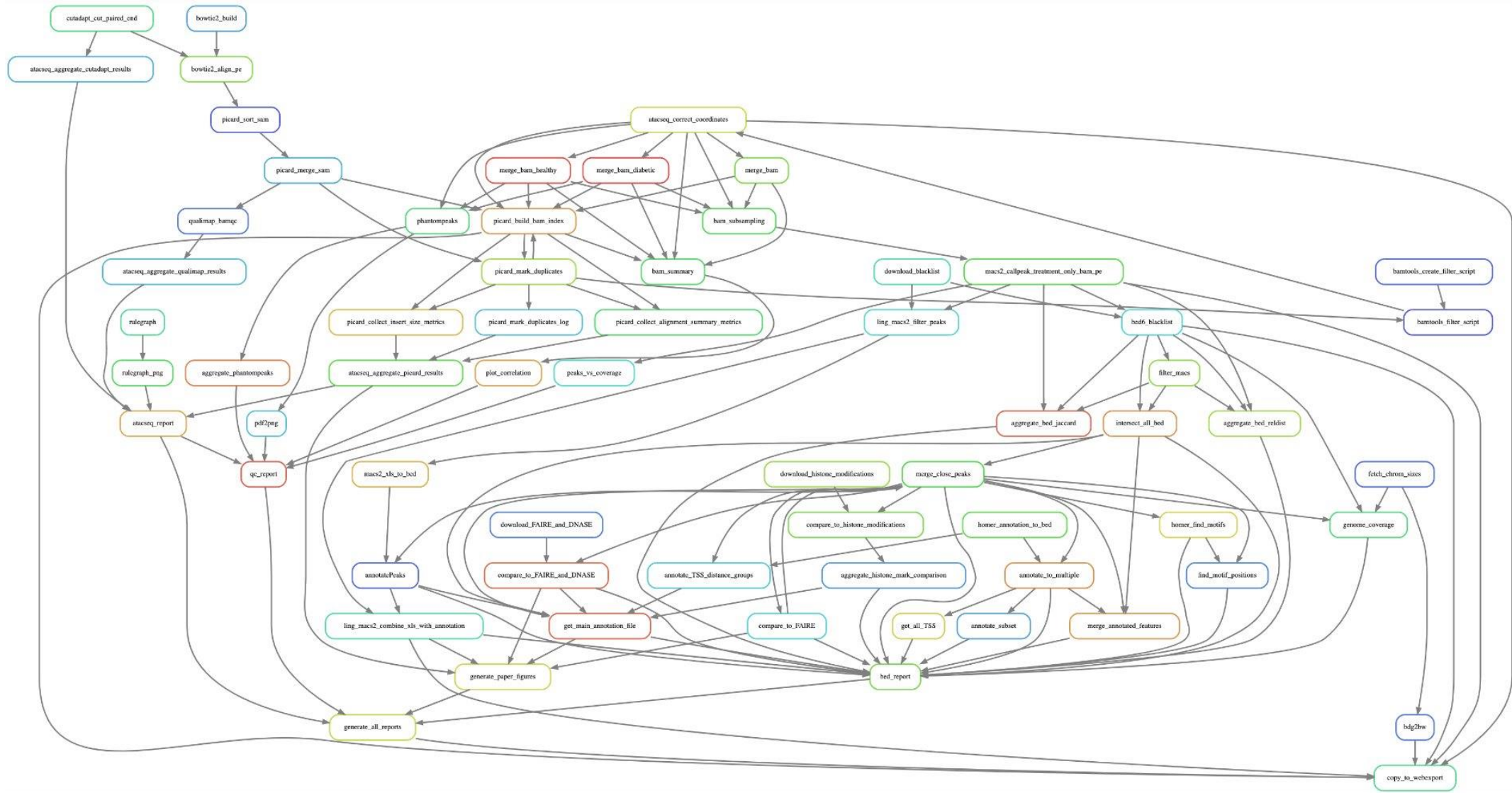
## Nextflow

Groovy
Can use both files and values
Each job has its own dir
“Push” - available inputs <code>Channel.fromPath('data/*.fastq')   map -&gt; process A -&gt; process B</code>
No
Yes (hashes processes, scripts code, input values). Changes invalidate the cache and trigger rerun
Forward (input - output)



# Scalability

Snakemake downside: A DAG can be **too big to hold in memory** or **too slow to process**



# Scalability

## Snakemake

Full DAG required?

✓ Yes

Asynchronous execution

✗ Limited

Scales to millions of tasks

⚠ Bottlenecks

Executor abstraction

Basic (manual) -pulgins

Memory-efficient scheduling

✗ Less so

## Nextflow

✗ No (dataflow channels)

✓ Yes

✓ Better suited

Rich & modular

✓ More scalable





 ~ 20 min

Now is YOUR time!

Nextflow exercises

- Build a pipeline from scratch and make it reproducible

If you have time, try the bonus exercise

**nf-core** 



`hds-sandbox.github.io/HPC-lab`

workshop >  
HPC Launch >  
Day 2 – Part 5 >  
Exercise I & II

# Problems/Issues/Comments

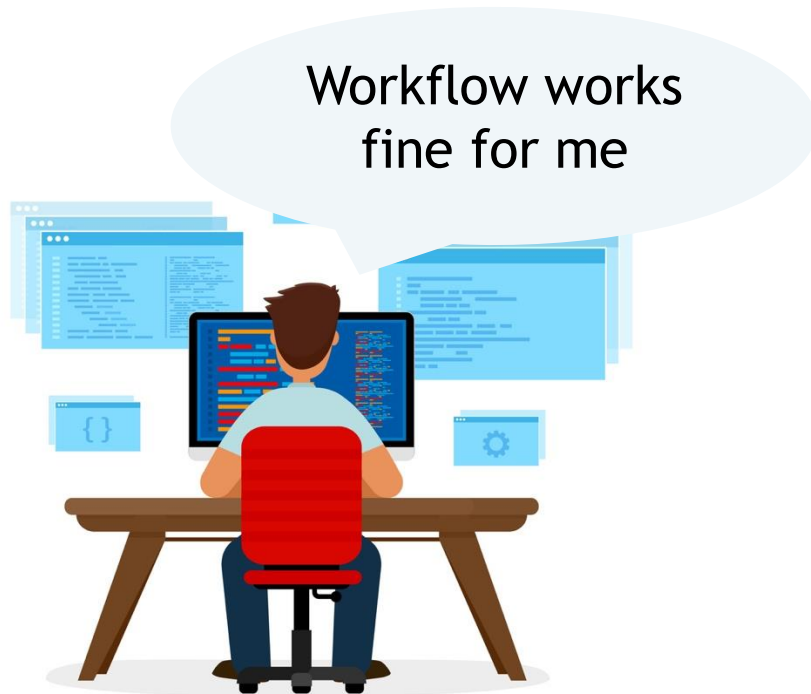
## Part II - Computations management

Workflow  
management  
systems

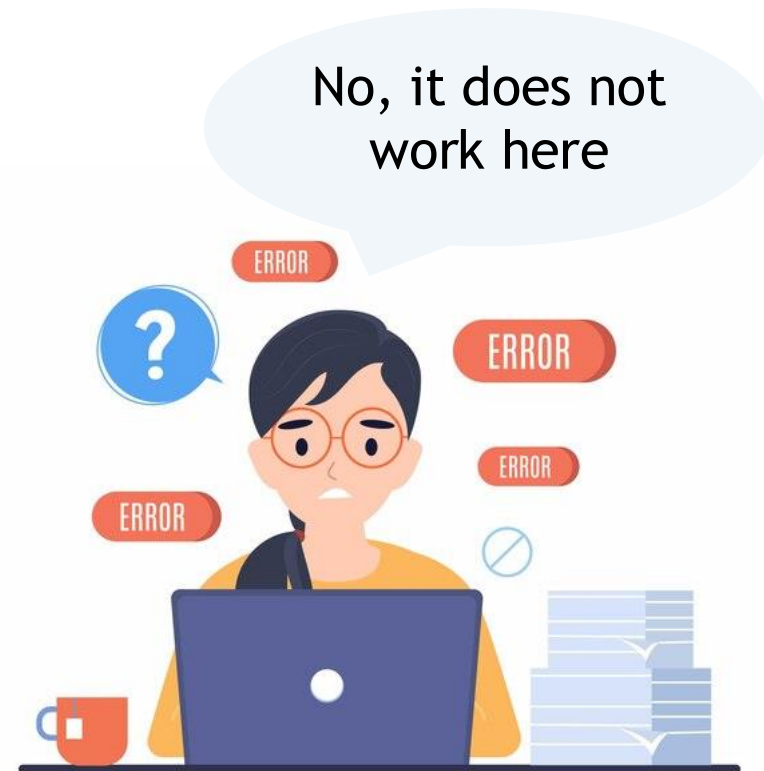
Integration with  
environment  
managers

# Why containers in workflows?

## Developer



## Tester/User



# Nextflow pipeline with containers

## Built-in integration with containers (and Conda)

### Advantages

- Maintainability
- Portability
- Reproducibility

### Popular containers

- Docker
- Singularity



```
nextflow run ... -with-singularity /path/to/image.img
```

```
custom.config
```

```
singularity {  
    enabled=true  
    process.container='shub://IARCBioinfo/nf_coverage_demo:v2.3'  
    pullTimeout='200 min'  
}
```



<https://www.nextflow.io/docs/latest/reference/process.html>

## module

[Environment Modules](#) is a package manager that allows you to dynamically configure your execution environment and easily switch between multiple versions of the same software tool.

If it is available in your system you can use it with Nextflow in order to configure the processes execution environment in your pipeline.

In a process definition you can use the `module` directive to load a specific module version to be used in the process execution environment. For example:

```
process basicExample {  
  module 'ncbi-blast/2.2.27'  
  
  ''''  
  blastp -query <etc..>  
  ''''  
}
```

You can repeat the `module` directive for each module you need to load. Alternatively multiple modules can be specified in a single `module` directive by separating all the module names by using a `:` (colon) character as shown below:





Documentation with good examples:

<https://www.nextflow.io/docs/latest/index.html>

Training material using GitHub codespaces:

<https://training.seqera.io/latest/#in-depth-nextflow-training>

Two specific examples for genomics and rnaseq:

- Genomics: [https://training.seqera.io/latest/nf4\\_science/genomics/](https://training.seqera.io/latest/nf4_science/genomics/)
- RNAseq: [https://training.seqera.io/latest/nf4\\_science/rnaseq/](https://training.seqera.io/latest/nf4_science/rnaseq/)



# Deploying nf-core pipelines on an HPC

- A community effort to collect curated set of analysis pipelines built using Nextflow
- Provides nice guidelines for >120 pipelines: <https://nf-co.re/pipelines>
- Each pipeline has its own documentation

## Input files

- .sbatch: file to wrap around the nextflow run command
- pipeline.conf: for defining **nextflow** resources
- pipeline\_params.yaml: for defining **nf-core** pipeline parameters
- Samplesheet.csv





# Nf-core template

```
>- > nf-core pipelines create
```

NF-CORE

Welcome to the nf-core pipeline creation wizard

This app will help you create a new Nextflow pipeline from the [nf-core/tools pipeline template](#).

The template helps anyone benefit from nf-core best practices, and is a requirement for nf-core pipelines.

💡 If you want to add a pipeline to nf-core, please join on Slack and discuss your plans with the community as early as possible; *ideally before you start on your pipeline!* See the [nf-core guidelines](#) and the [#new-pipelines](#) Slack channel for more information.

Let's go!

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

python3.12 - nf-core

nf-core pipelines create - Create a new pipeline with the nf-core pipeline template

Choose pipeline type

Choose "nf-core" if:

- You want your pipeline to be part of the nf-core community
- You think that there's an outside chance that it ever *could* be part of nf-core

nf-core

Choose "Custom" if:

- Your pipeline will *never* be part of nf-core
- You want full control over *all* features that are included from the template (including those that are mandatory for nf-core).

Custom

What's the difference?

Choosing "nf-core" effectively pre-selects the following template features:

- GitHub Actions continuous-integration configuration files:
  - Pipeline test runs: Small-scale (GitHub) and large-scale (AWS)
  - Code formatting checks with Prettier
  - Auto-fix linting functionality using [@nf-core-bot](#)
  - Marking old issues as stale
- Inclusion of [shared nf-core configuration profiles](#)

# HPC-Pipes is complete

