Swiss Institute of Bioinformatics

Nextflow in Action:

Build Smarter, Faster, Reproducible Pipelines

PhDc Jeferyd Yepes-García
University of Fribourg

BUGFri

March 17th, 2026

elixir SWITZERLAND

Swiss Institute of Bioinformatics

About Me

nextflow AMBASSADOR

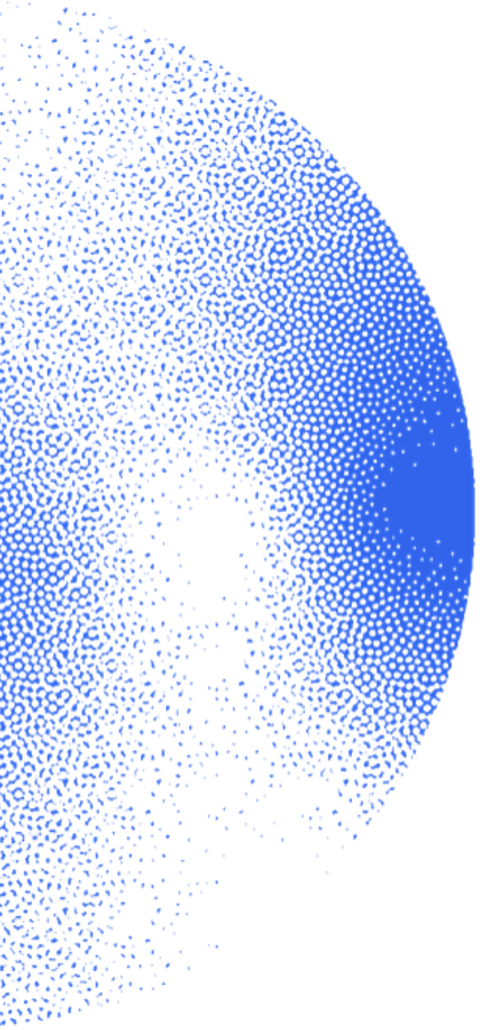
TaxoFlow The tutorial

nf-core/mag

MAGFlow Workflow for MAG quality assessment and taxonomical annotation

elixir SWITZERLAND

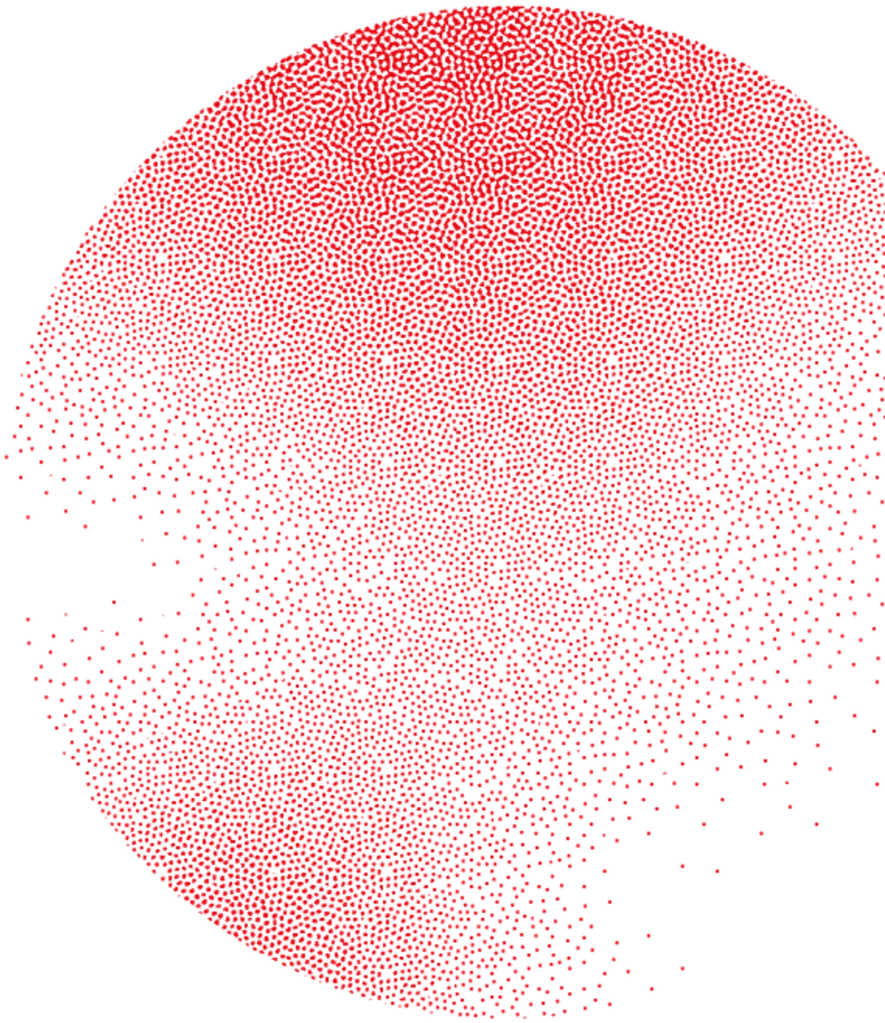
Course schedule



Session	Start	End	Topic
Session 1	09:00	09:30	Workflow managers and Nextflow architecture
Exercises 1	09:30	09:50	Intro pipeline
Session 2	09:50	10:15	Channels, modules, parameters and variables
	10:15	10:30	Break
Exercises 2	10:30	11:30	Hello Pipeline
Session 3	11:30	12:00	Operators, executors, software management and profiles
	12:00	13:00	Lunch break
Exercises 3	13:00	13:45	RNA-seq pipeline
Session 4	13:45	14:15	Conditionals, scripts *à la carte*, report
Exercises 4	14:15	15:00	Metagenomics pipeline
	15:00	15:15	Break
Exercises 5	15:15	16:30	Wrap your pipeline or genomics pipeline or genomics pipeline
Session 5	16:30	16:45	nf-core
Q&A	16:45	17:00	Wrap-up

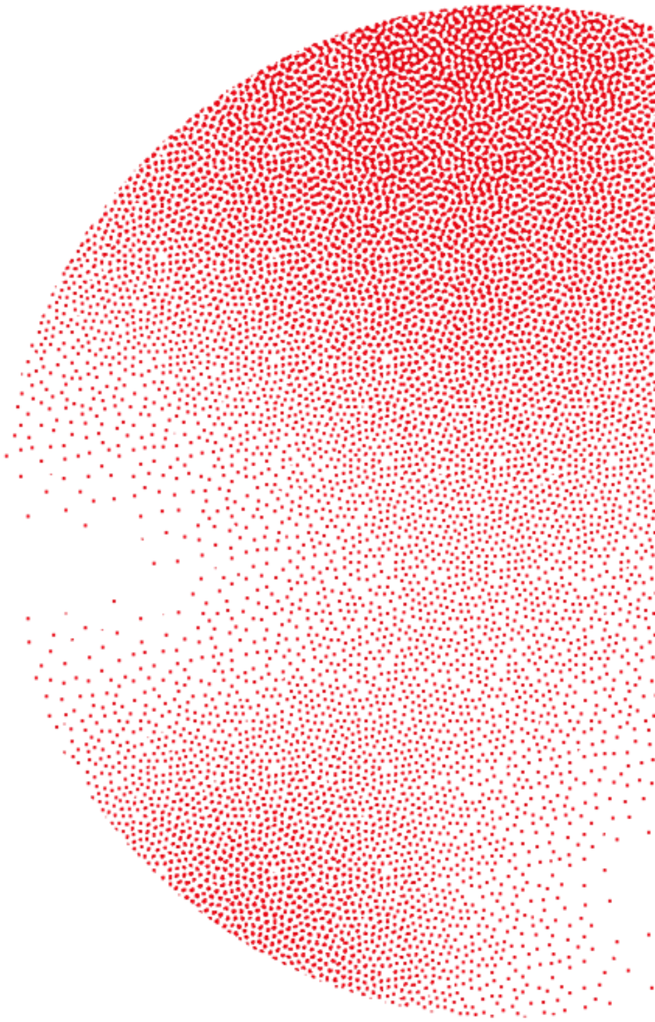
This presentation

- Learning outcomes
- Context
- Why Nextflow? (and not others)
- Language generalities
- Dataflow paradigm
- Processes
- Workflows



Learning outcomes

- Recognize the benefits of wrapping your routine analysis with a workflow orchestrator.
- Learn how to modularize their pipelines to enhance the control execution.
- Develop your own pipelines wrapped with Nextflow.
- Run your pipelines in different environments.



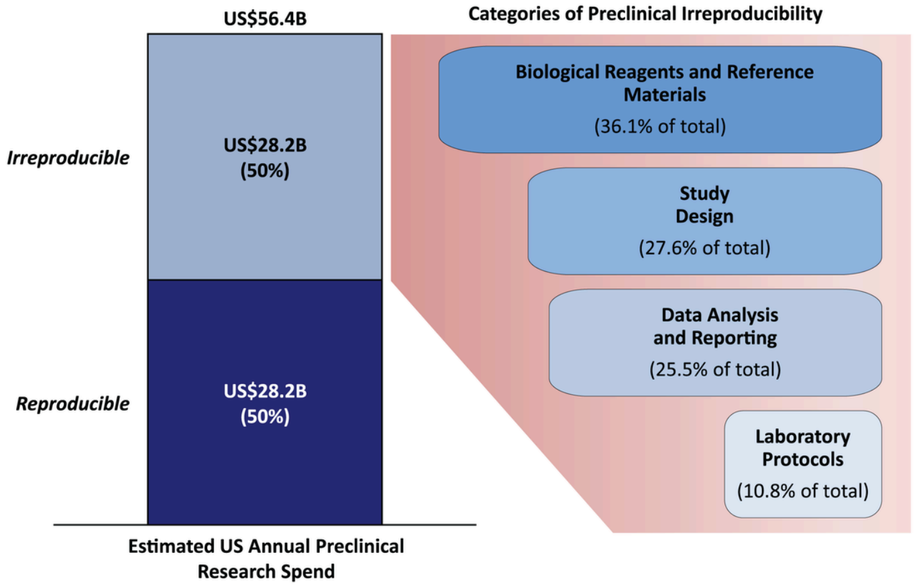
Reproducibility crisis?

PERSPECTIVE

The Economics of Reproducibility in Preclinical Research

Leonard P. Freedman[1]*, Iain M. Cockburn[2], Timothy S. Simcoe[2,3]

1 Global Biological Standards Institute, Washington, D.C., United States of America, **2** Boston University School of Management, Boston, Massachusetts, United States of America, **3** Council of Economic Advisers, Washington, D.C., United States of America



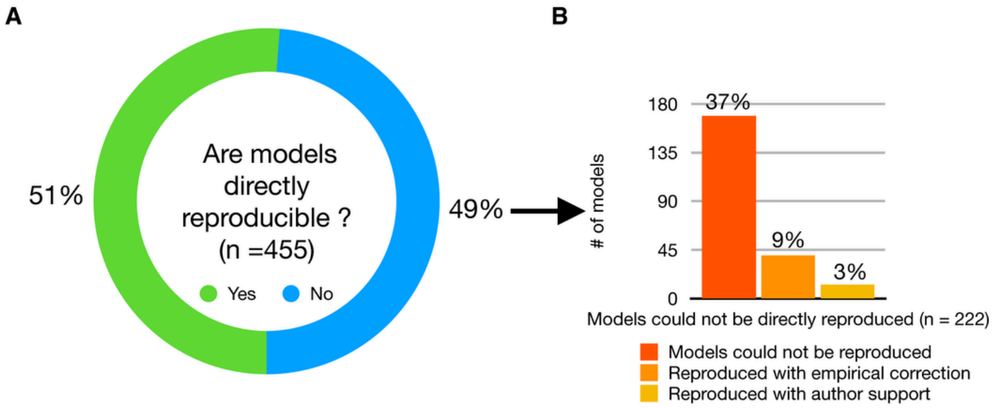
Commentary

OPEN ACCESS

molecular systems biology

Reproducibility in systems biology modelling

Krishna Tiwari[1,2], Sarubini Kananathan[1], Matthew G Roberts[1], Johannes P Meyer[1], Mohammad Umer Sharif Shohan[1], Ashley Xavier[1], Matthieu Maire[1], Ahmad Zyoud[1], Jinghao Men[1], Szeyi Ng[1], Tung V N Nguyen[1], Mihai Glont[1], Henning Hermjakob[1,3,*] & Rahuman S Malik-Sheriff[1,**]



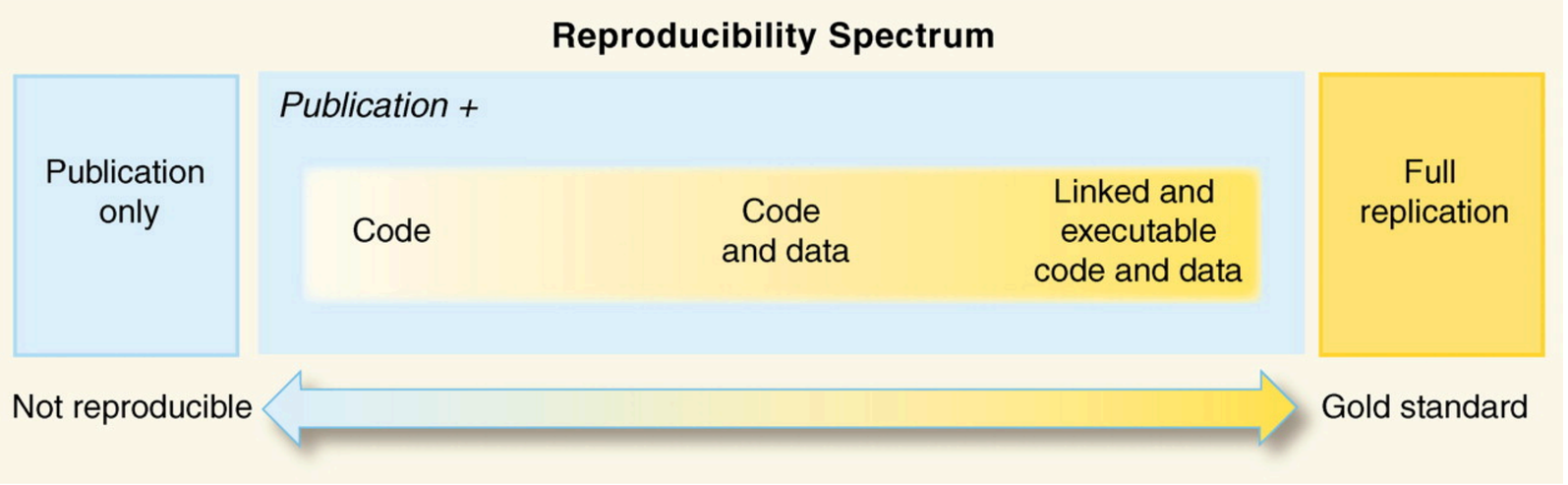
Reproducibility in computational biology and bioinformatics

TABLE 1 Simple grid-based system for defining concepts that can be used to describe the validity of a result[a]

Methods	Same experimental system	Different experimental system
Same methods	Reproducibility	Replicability
Different methods	Robustness	Generalizability

[a]This is a generalization of the approach used by Whitaker (9), who used it to describe computational analyses.

Schloss (2018)



Peng (2011)

Reproducibility in computational biology and bioinformatics

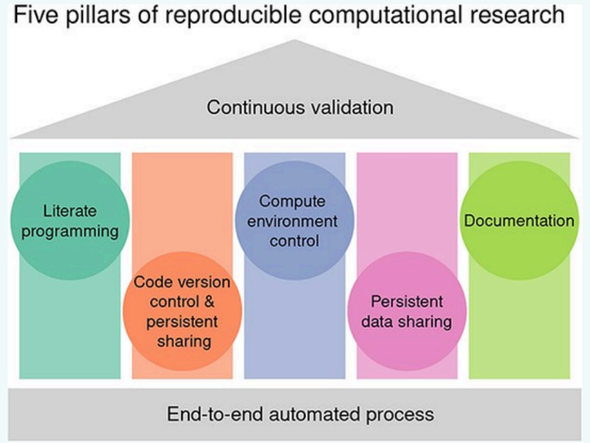
In computational biology and bioinformatics, reproducibility implicitly means the ability to obtain **identical outputs** from the **same code, data, and software environment** (Ziemann et al. 2023).

```
Output = Algorithm (Code) + Data + Parameters
```

Problems

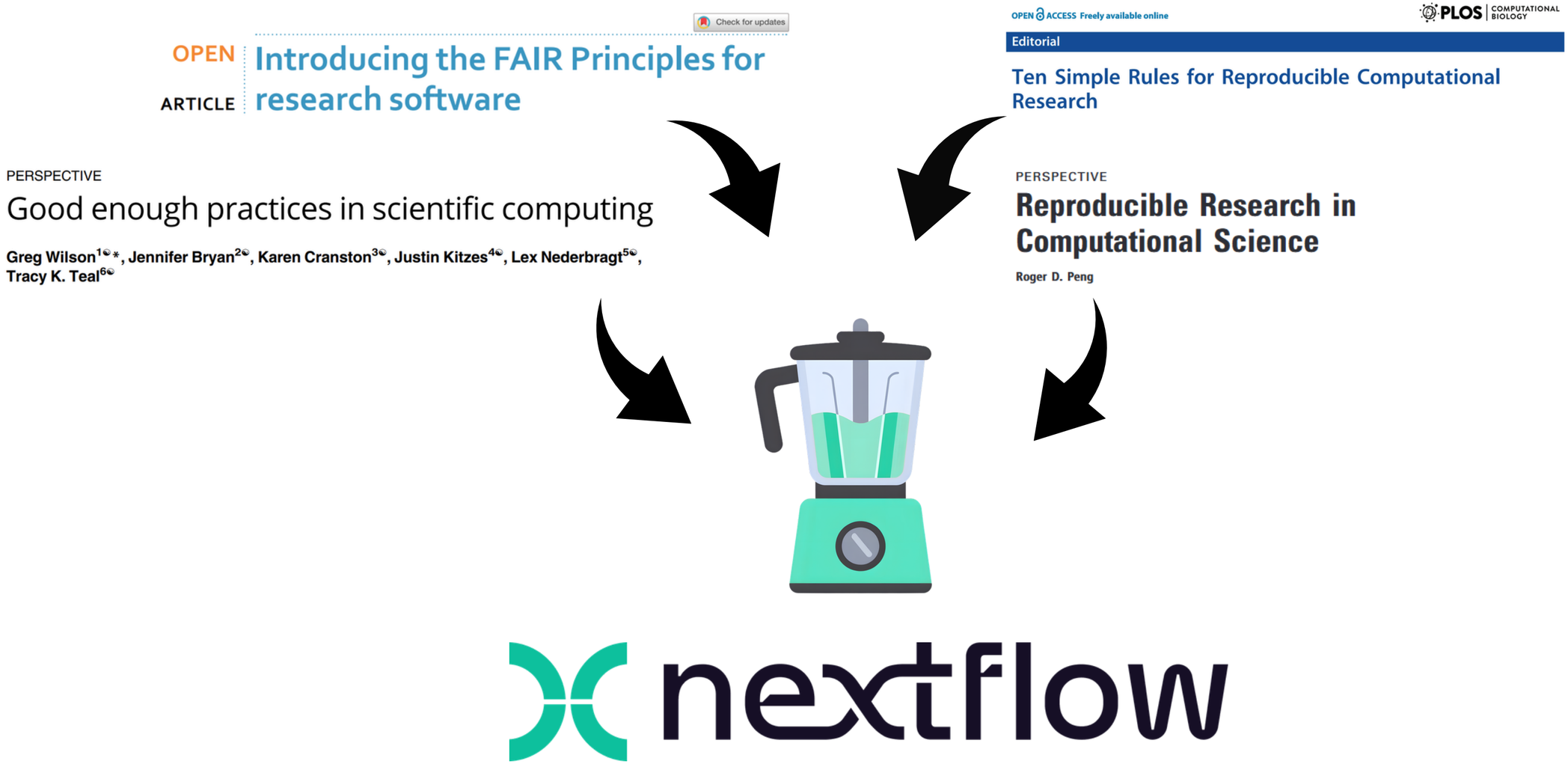
- Versioning.
- Data availability.
- Environment.
- Parameters setting.
- Hidden manipulation steps.
- Random seeds.
- ...

Solutions



Ziemann et al. (2023)

Reproducibility in computational biology and bioinformatics



What is Nextflow?

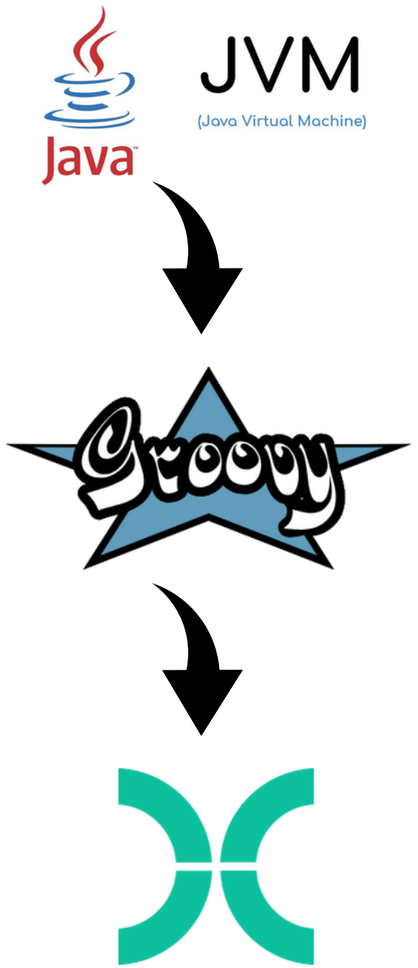
Correspondence | Published: 11 April 2017

Nextflow enables reproducible computational workflows

Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo & Cedric Notredame

Features

- Domain Specific Language (currently DSL2)
- Dataflow programming model
- Parallel and scalable execution
- Reproducible
- Portable
- Workflow Graph Construction (DAG)

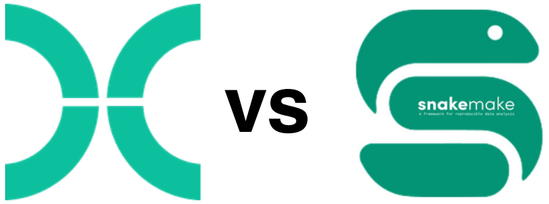


It’s not the only option

Table 1 | Overview of workflow managers for bioinformatics (top, editable version; bottom, image version)

Tool	Class	Ease of use[a]	Expressiveness[b]	Portability[c]	Scalability[d]	Learning resources[e]	Pipeline initiatives[f]
Galaxy	Graphical	●●●	●○○	●●●	●●●	●●●	●●○
KNIME	Graphical	●●●	●○○	○○○	●●◐	●●●	●●○
Nextflow	DSL	●●○	●●●	●●●	●●●	●●●	●●●
Snakemake	DSL	●●○	●●●	●●◐	●●●	●●○	●●●
GenPipes	DSL	●●○	●●●	●●○	●●○	●●○	●●○
bPipe	DSL	●●○	●●●	●●○	●●◐	●●○	●○○
Pachyderm	DSL	●●○	●●●	●○○	●●○	●●●	○○○
SciPipe	Library	●●○	●●●	○○○	○○○	●●○	○○○
Luigi	Library	●●○	●●●	●○○	●●◐	●●○	○○○
Cromwell + WDL	Execution + workflow specification	●○○	●●○	●●●	●●◐	●●○	●●○
cwltool + CWL	Execution + workflow specification	●○○	●●○	●●◐	○○○	●●●	●●○
Toil + CWL/ WDL/Python	Execution + workflow specification	●○○	●●●	●◐○	●●●	●●○	●●○

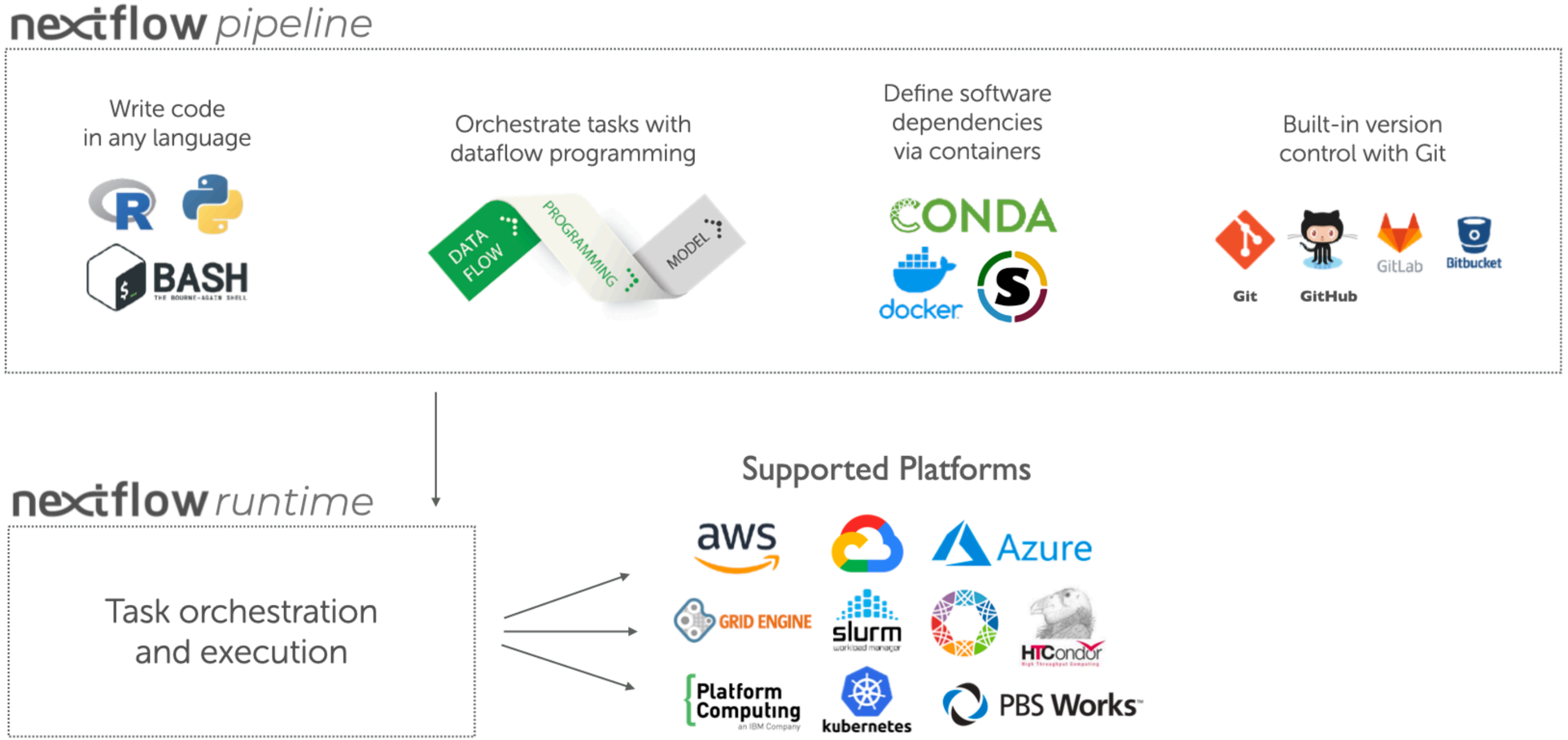
Wratten et al. (2021)



Feature	Nextflow	Snakemake
Language	Groovy-based DSL	Python-based syntax
Ease of Use	Steep learning curve	Easier for Python users
Parallel Execution	Excellent (dataflow model)	Good (dependency graph)
Scalability	High (supports cloud, HPC, containers)	Moderate (limited native cloud support)
Containerization	Supports Docker, Singularity, Conda	Supports Docker, Singularity, Conda
Cloud Support	Built-in AWS, Google Cloud, Azure	Needs additional tools for cloud usage
Reproducibility	Strong (workflow versioning)	Strong (containerized environments)
Pipeline Registry	nf-core (90+ pipelines)	Snakemake Catalog

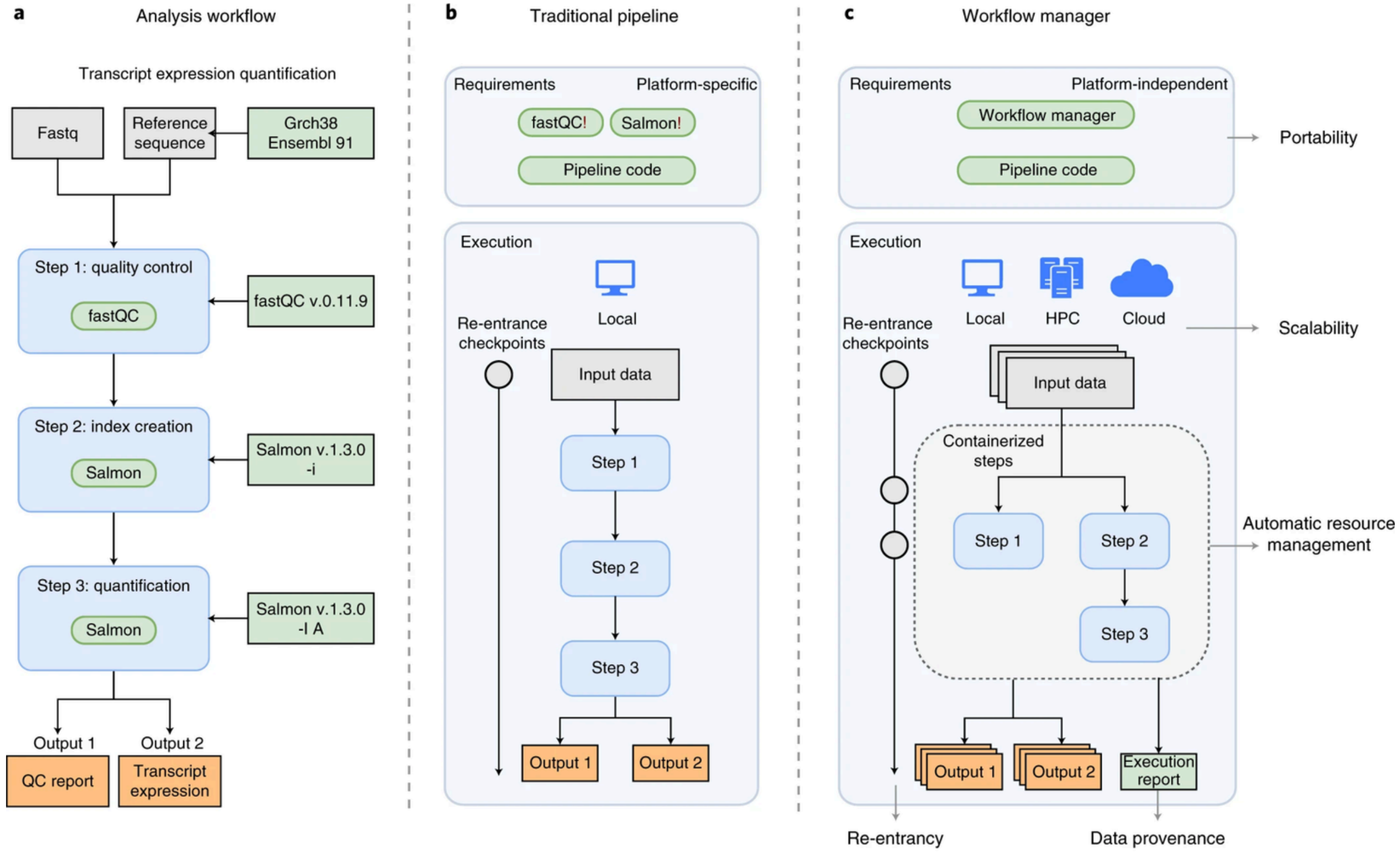
Source: Tasrie IT Services

Nextflow's execution abstraction: *Write once, run anywhere*



Source: [Seqera]

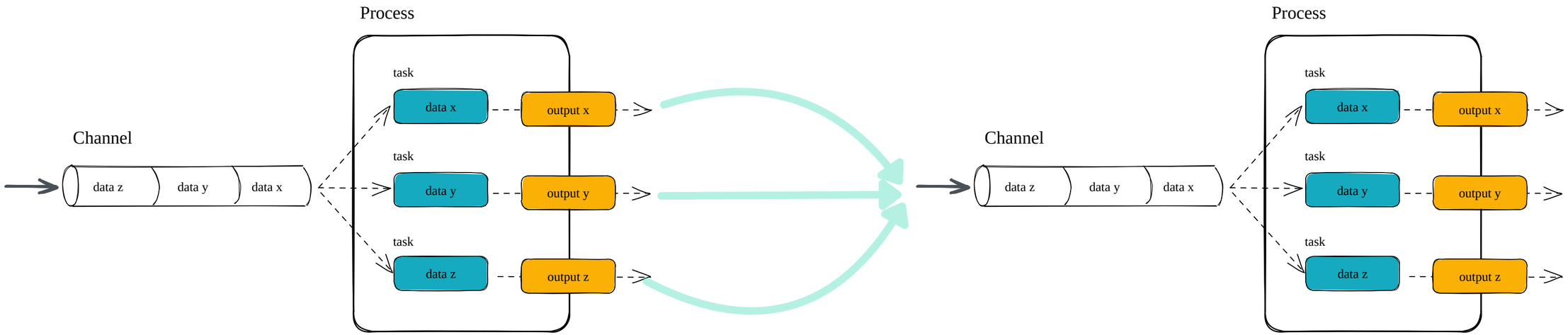
Real pipeline



Wratten et al. (2021)

Input data | Output data | Software, versions, parameters | ! Fixed version, local compute environment

Dataflow paradigm: *The output from one process is the input for the following*



Main advantages

Asynchronous channels.
There is no communication among data.
Parallelism becomes an inherent property.
Processes run in isolation.

The basic unit: *process*

Command

```
~$ echo "Hello world!" > hello.txt
```

Process

```
process hello_world {
    output:
        path 'hello.txt'
    script:
        '''
        echo "Hello world!" > hello.txt
        '''
}
```

Definitions

output → What the process is generating.

script → What the process is executing.

Where is the input?

Dataflow paradigm: *The output from one process is the input for the following*

Output qualifiers

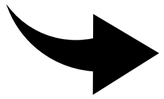
val → Any variable value (string, float, etc.)
path → Any file (fasta, bam, vcf, etc.) or directory
stdout → Command line output
tuple → A combination of the others.
...

Input qualifiers

Dataflow paradigm: *The output from one process is the input for the following*

Output qualifiers

val → Any variable value (string, float, etc.)
path → Any file (fasta, bam, vcf, etc.) or directory
stdout → Command line output
tuple → A combination of the others.
...



Output qualifiers

val → Any variable value (string, float, etc.)
path → Any file (fasta, bam, vcf, etc.) or directory
stdout → Command line output
tuple → A combination of the others.
...

Dataflow paradigm: *The output from one process is the input for the following*

```
process MULTIQC {

    input:
    path '*'
    val output_name

    output:
    path "${output_name}.html", emit: report
    path "${output_name}_data", emit: data

    script:
    """
    multiqc . -n ${output_name}.html
    """
}
```

Important

Multiple inputs/outputs
Use of wildcards
Dynamic file naming (variable expansion)
Optional: emit (eases file handling)

Question

If a subsequent process is taking the output of this process, how the input would look like?

Real process

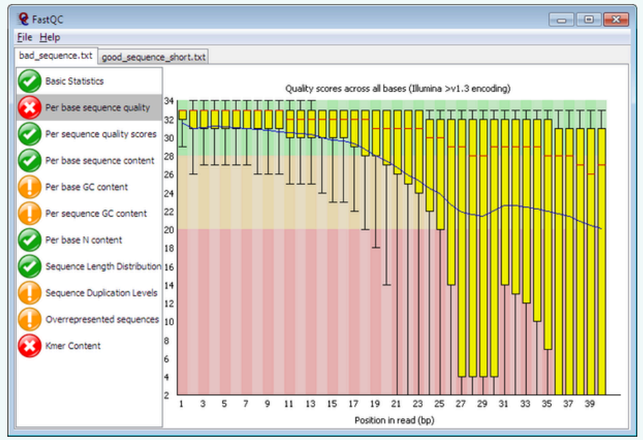
Set of reads

```
ENCSR000COQ1_1.fastq.gz
ENCSR000COQ1_2.fastq.gz
ENCSR000COQ2_1.fastq.gz
ENCSR000COQ2_2.fastq.gz
ENCSR000COR1_1.fastq.gz
ENCSR000COR1_2.fastq.gz
ENCSR000COR2_1.fastq.gz
ENCSR000COR2_2.fastq.gz
ENCSR000CPO1_1.fastq.gz
ENCSR000CPO1_2.fastq.gz
ENCSR000CPO2_1.fastq.gz
ENCSR000CPO2_2.fastq.gz
```

Command

```
$ fastqc -o fastqc *.fastq.gz
```

Report



In other words:

nextflow

Input → **Process** → **Output**

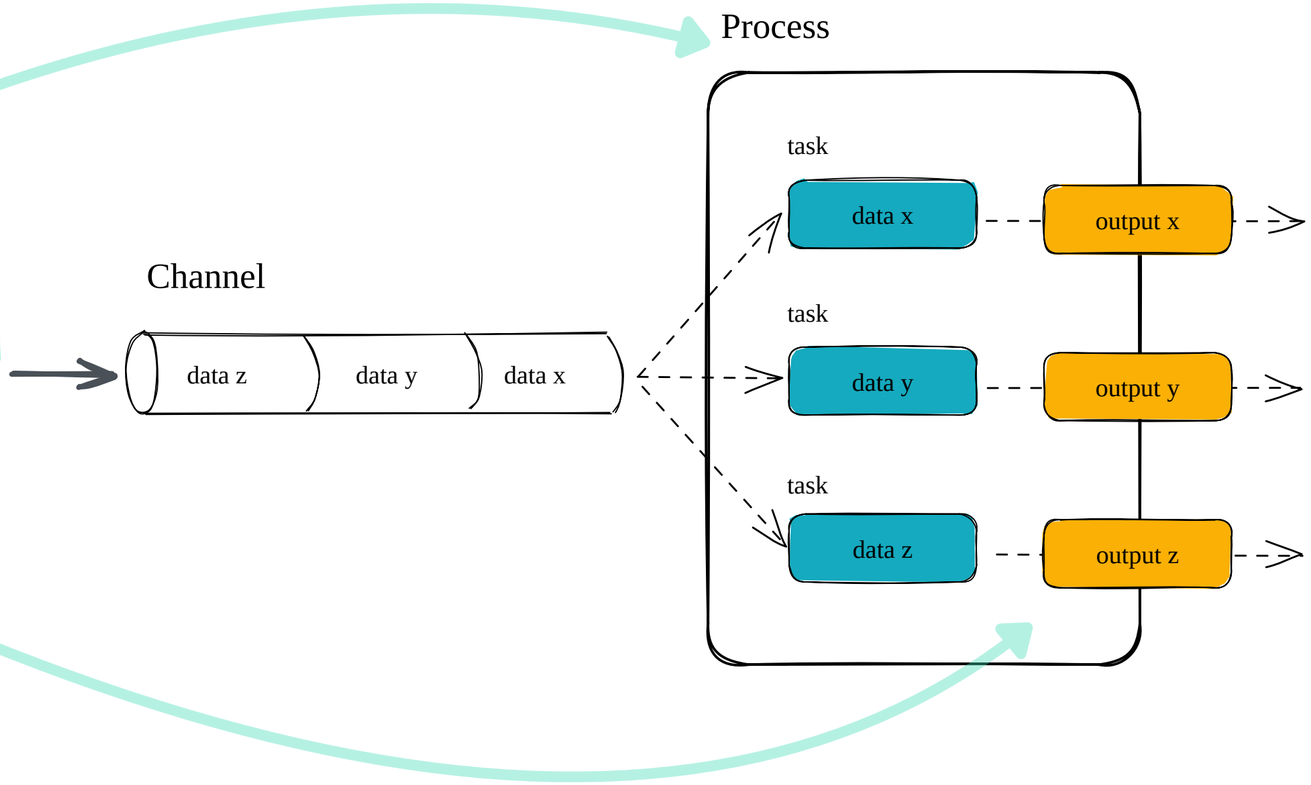
What’s happening under the hood?

```
process FASTQC {

    input:
    path reads

    output:
    path "*_fastqc.zip", emit: zip
    path "*_fastqc.html", emit: html

    script:
    """
    fastqc ${reads}
    """
}
```



Executing the process(es): *workflow*

Process

```
process hello_world {
    output:
        path 'hello.txt'
    script:
        '''
        echo "Hello world!" > hello.txt
        '''
}
```

Workflow

```
workflow {

    main:
    hello_world()

}
```

Definitions

main → Processes being executed.

Executing the process(es): *workflow*

Workflow

```
workflow {

    main:
    sayHello(greeting_ch)
    convertToUpper(sayHello.out)

    publish:
    first_output = sayHello.out
    uppercased = convertToUpper.out
}

output {
    first_output {
        path 'hello_workflow'
        mode 'copy'
    }
    uppercased {
        path 'hello_workflow'
        mode 'copy'
    }
}
```

Important

Order of execution
Input/output must match
What you want to publish or produce as output of the workflow

Definitions

publish → The files or values to be placed in the output.

output → Where the files are going to be stored.

Executing the process(es): *workflow*

Workflow

```
workflow {

    main:
    sayHello(greeting_ch)
    convertToUpper(sayHello.out)

    publish:
    first_output = sayHello.out
    uppercased = convertToUpper.out
}

output {
    first_output {
        path 'hello_workflow'
        mode 'copy'
    }
    uppercased {
        path 'hello_workflow'
        mode 'copy'
    }
}
```

Start of the workflow

Following process

Output from the previous process. **It's a channel!**

Which files will be published

Directory to place each output

Important

Order of execution

Input/output must match

What you want to publish or produce as output of the workflow

Definitions

publish → The files or values to be placed in the output.

output → Where the files are going to be stored.

It’s your time!

Please, before start working, update the directory on GitHub Codespaces:

```
cd /workspaces/nextflow-training
git fetch origin
git reset --hard origin/main
```