

Introduction to Bioinformatics workflows with Nextflow and nf-core

John A. Juma

28th April, 2022

j.juma@cgiar.org



Overview

Questions

1. What is a workflow and what are workflow management systems?
2. Why should I use a workflow management system?
3. What is Nextflow?
4. What are the main features of Nextflow?
5. What are the main components of a Nextflow script?
6. How do I run a Nextflow script?

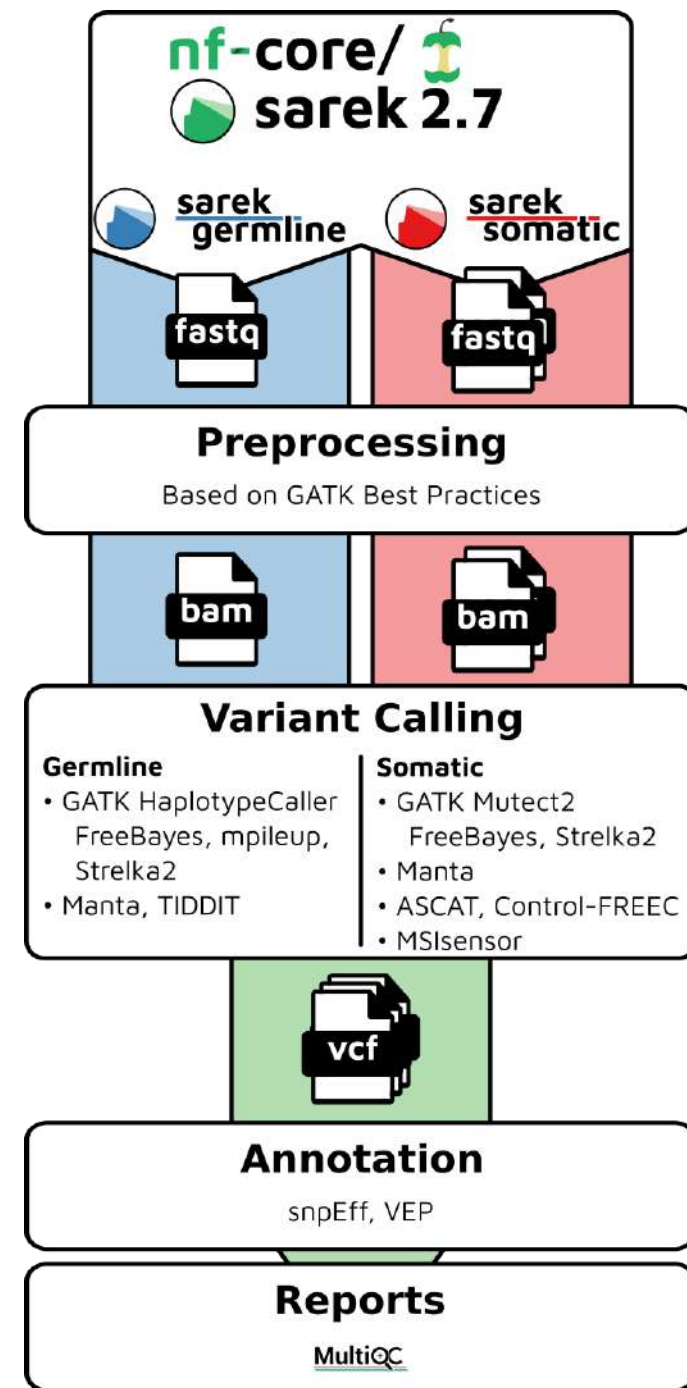
Objectives

1. Understand what a workflow management system is.
2. Understand the benefits of using a workflow management system.
3. Explain the benefits of using Nextflow as part of your bioinformatics workflow.
4. Explain the components of a Nextflow script.
5. Run a Nextflow script.

Workflows

Analysing data involves a sequence of tasks, including gathering, cleaning, and processing data. These sequence of tasks are called a **workflow** or a **pipeline**.

Example bioinformatics variant calling workflow/pipeline diagram from nf-core (<https://nf-co.re/sarek>)



Workflow management systems

Workflow Management Systems (WfMS), such as Snakemake, Galaxy, and Nextflow have been developed specifically to manage computational data-analysis workflows in fields such as Bioinformatics, Imaging, Physics, and Chemistry.

WfMS contain multiple features that simplify the development, monitoring, execution and sharing of pipelines. Key features include;

1. **Run time management:** Management of program execution on the operating system and splitting tasks and data to run at the same time in a process called parallelization.
2. **Software management:** Use of technology like containers, such as [Docker](#) or [Singularity](#), that packages up code and all its dependencies so the application runs reliably from one computing environment to another.
3. **Portability & Interoperability:** Workflows written on one system can be run on another computing infrastructure e.g., local computer, compute cluster, or cloud infrastructure.
4. **Reproducibility:** The use of software management systems and a pipeline specification means that the workflow will produce the same results when re-run, including on different computing platforms.
5. **Reentrancy:** Continuous checkpoints allow workflows to resume from the last successfully executed steps.

Differences between Nextflow and Snakemake

	Snakemake	Nextflow
Language	Python	Groovy
Data	Everything is a file	Can use both files and values
Execution	Working directory	Each job in its own directory
Philosophy	“Pull”	“Push”
Dry runs	Yes	No
Track code changes	No	Yes

Scripting language

- Nextflow scripts are written using a language intended to simplify the writing of workflows. Languages written for a specific field are called **Domain Specific Languages (DSL)**, e.g., SQL is used to work with databases, and AWK is designed for text processing.
- In practical terms the Nextflow scripting language is an extension of the [Groovy programming language](#), which in turn is a super-set of the Java programming language. Groovy simplifies the writing of code and is more approachable than Java. Groovy semantics (syntax, control structures, etc) are documented [here](#).
- The approach of having a simple DSL built on top of a more powerful general purpose programming language makes Nextflow very flexible. The Nextflow syntax can handle most workflow use cases with ease, and then Groovy can be used to handle corner cases which may be difficult to implement using the DSL.

DSL2 syntax

Nextflow (version > 20.07.1) provides a revised syntax to the original DSL, known as DSL2. The DSL2 syntax introduces several improvements such as **modularity** (separating components to provide flexibility and enable reuse), and improved **data flow manipulation**. This further simplifies the writing of complex data analysis pipelines, and enhances workflow readability, and reusability.

This feature is enabled by the following directive at the beginning a workflow script:

```
nextflow.enable.dsl=2
```

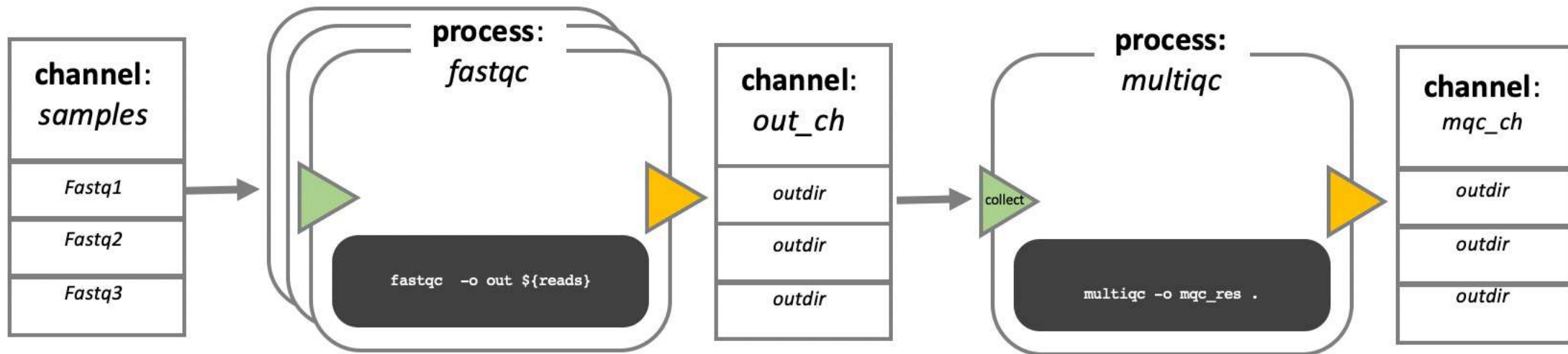
Processes, channels, and workflows

Nextflow workflows have three main parts; **processes**, **channels**, and **workflows**.

- Processes describe a task to be run. A process script can be written in any scripting language that can be executed by the Linux platform (Bash, Perl, Ruby, Python, etc.). Processes spawn a task for each complete input set. Each task is executed independently and cannot interact with another task. The only way data can be passed between process tasks is via **asynchronous queues**, called channels. Processes define inputs and outputs for a task.
- Channels are then used to manipulate the flow of data from one process to the next. The interaction between processes, and ultimately the pipeline execution flow itself, is then explicitly defined in a workflow section.

Processes and channels

In the following example we have a channel containing three elements, e.g., 3 data files. We have a process that takes the channel as input. Since the channel has three elements, three independent instances (tasks) of that process are run in parallel. Each task generates an output, which is passed to another channel and used as input for the next process.

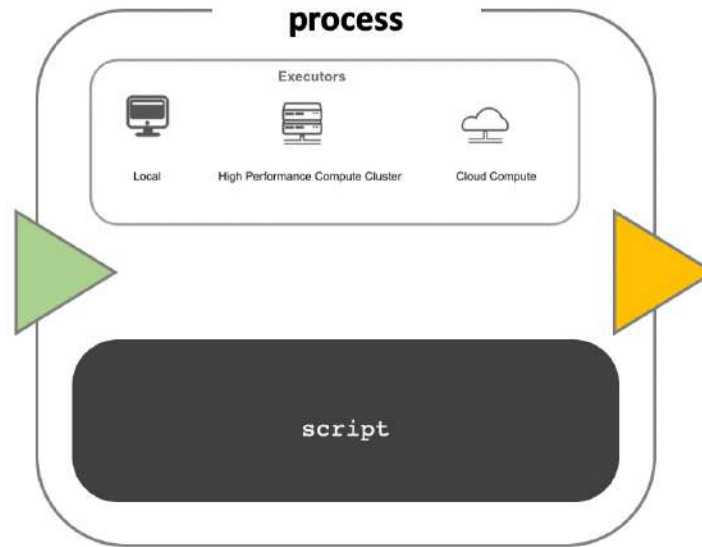


Nextflow process flow diagram

Workflow execution

While a process defines what command or script has to be executed, the executor determines how that script is run in the target system.

If not otherwise specified, processes are executed on the local computer. The local executor is very useful for pipeline development, testing, and small scale workflows, but for large scale computational pipelines, a High Performance Cluster (HPC) or Cloud platform is often required.



Nextflow Executors

Process definition block

```
nextflow.enable.dsl=2
```

```
process < NAME > {  
    [ directives ]
```

```
    input:
```

```
    < process inputs >
```

```
    output:
```

```
    < process outputs >
```

```
    when:
```

```
    < condition >
```

```
    [script|shell|exec]:
```

```
    < user script to be executed >
```

```
}
```

Run a command in shell

```
process PROCESSBAM {
```

```
    script:
```

```
    """
```

```
    samtools sort -o ref1.sorted.bam ${projectDir}/data/yeast/bams/ref1.bam
```

```
    samtools index ref1.sorted.bam
```

```
    samtools flagstat ref1.sorted.bam
```

```
    """
```

```
}
```

Run a python script

```
//process_python.nf  
nextflow.enable.dsl=2
```

```
process PYSTUFF {
```

```
    script:
```

```
    """
```

```
    #!/usr/bin/env python
```

```
    import gzip
```

```
    reads = 0
```

```
    bases = 0
```

```
    with gzip.open('${projectDir}/data/yeast/reads/ref1_1.fq.gz', 'rb') as read:
```

```
        for id in read:
```

```
            seq = next(read)
```

```
            reads += 1
```

```
            bases += len(seq.strip())
```

```
            next(read)
```

```
            next(read)
```

```
    print("reads", reads)
```

```
    print("bases", bases)
```

```
    """
```

```
}
```

```
workflow {
```

```
    PYSTUFF()
```

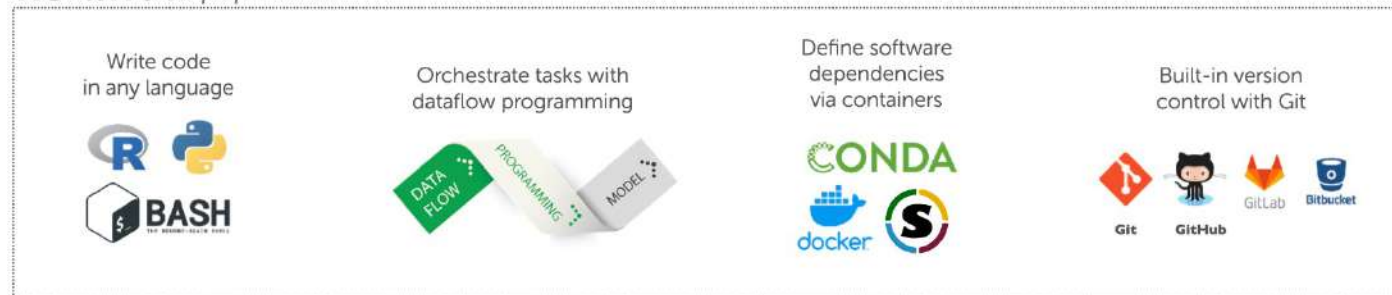
```
}
```

Your first script

1. An optional interpreter directive (“Shebang”) line, specifying the location of the Nextflow interpreter.
`nextflow.enable.dsl=2` to enable DSL2 syntax.
2. A multi-line Nextflow comment, written using C style block comments, followed by a single line comment.
3. A pipeline parameter `params.input` which is given a default value, of the relative path to the location of a compressed fastq file, as a string.
4. An unnamed `workflow` execution block, which is the default workflow to run.
5. A Nextflow `channel` used to read in data to the workflow.
6. A call to the process `NUM_LINES`. An operation on the process output, using the channel operator `view()`.
7. A Nextflow process block named `NUM_LINES`, which defines what the `process` does.
8. An input definition block that assigns the input to the variable `read`, and declares that it should be interpreted as a file `path`.
9. An output definition block that uses the Linux/Unix standard output stream `stdout` from the script block.

Summary

nextflow pipeline



nextflow runtime

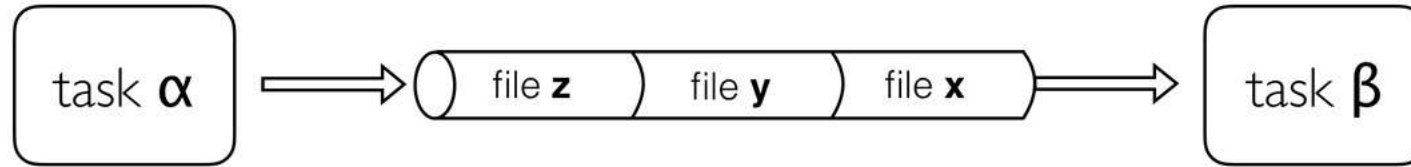


Supported Platforms



Channels and Operators

Channels



Channels are **asynchronous**, which means that outputs from a set of processes will not necessarily be produced in the same order as the corresponding inputs went in. However, the first element into a channel queue is the first out of the queue (**First in - First out**). This allows processes to run as soon as they receive input from a channel. Channels only send data in one direction, from a producer (a process/operator), to a consumer (another process/operator).

Channel types

Nextflow distinguishes between two different kinds of channels: **queue** channels and **value** channels.

Queue channel

Queue channels are a type of channel in which data is consumed (used up) to make input for a process/operator. Queue channels can be created in two ways:

1. As the outputs of a process.
2. Explicitly using channel factory methods such as [Channel.of](#) or [Channel.fromPath](#).

Value channels

The second type of Nextflow channel is a value channel. A **value** channel is bound to a **single** value (***singleton***). A value channel can be used an unlimited number of times since its content is not consumed. This is also useful for processes that need to reuse input from a channel, for example, a reference genome sequence file that is required by multiple steps within a process, or by more than one process.

In Nextflow DSL1 queue channels can only be used once in a workflow, either connecting workflow input to process input, or process output to input for another process. In DSL2 we can use a queue channel multiple times.

Channel types

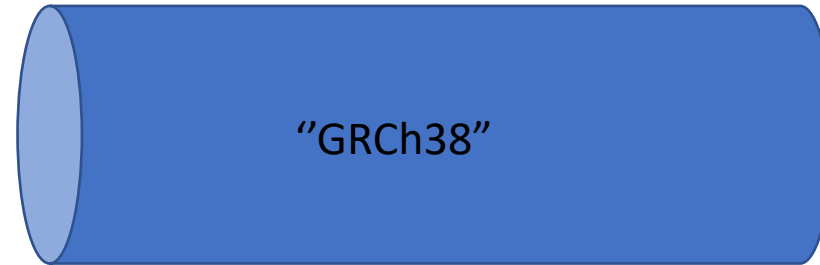
Queue channel factory

Queue (consumable) channels can be created using the following channel factory methods.

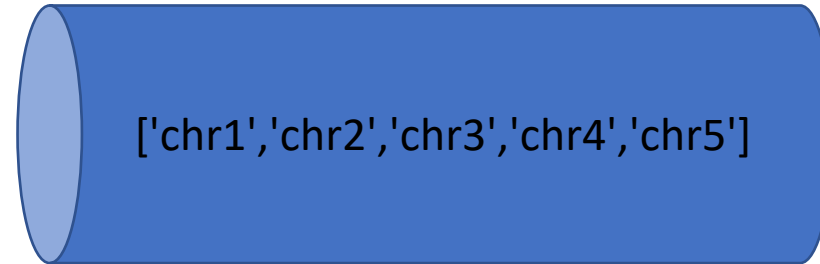
- `Channel.of`
- `Channel.fromList`
- `Channel.fromPath`
- `Channel.fromFilePairs`
- `Channel.fromSRA`

Operators

```
ch1 = Channel.value('GRCh38')
```



```
ch2 = Channel.value( ['chr1','chr2','chr3','chr4','chr5'] )
```

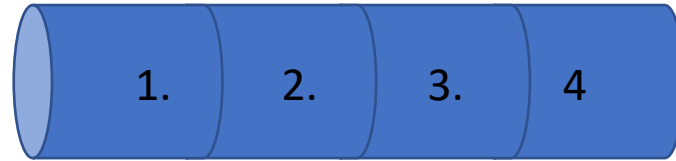


```
chromosome_ch = Channel.of( 'chr1','chr3','chr5','chr7' )
```

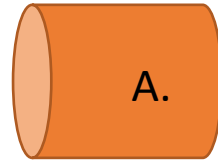


ch1.mix(ch2).collect()

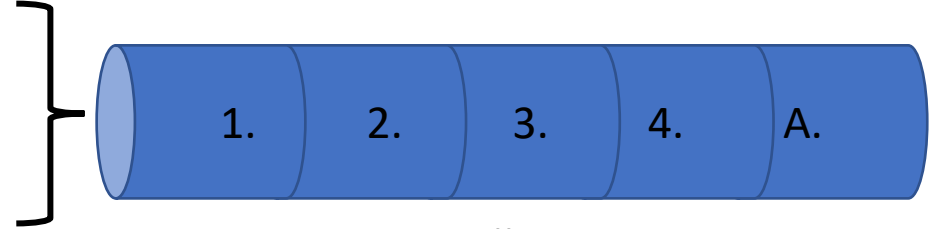
ch1=channel.of(1,2,3,4)



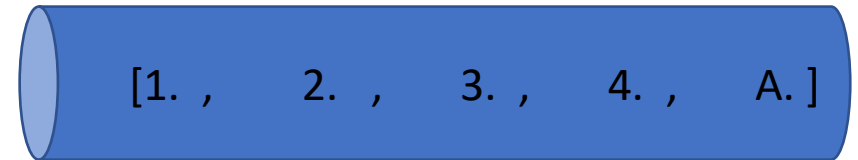
ch2=channel.of('A')



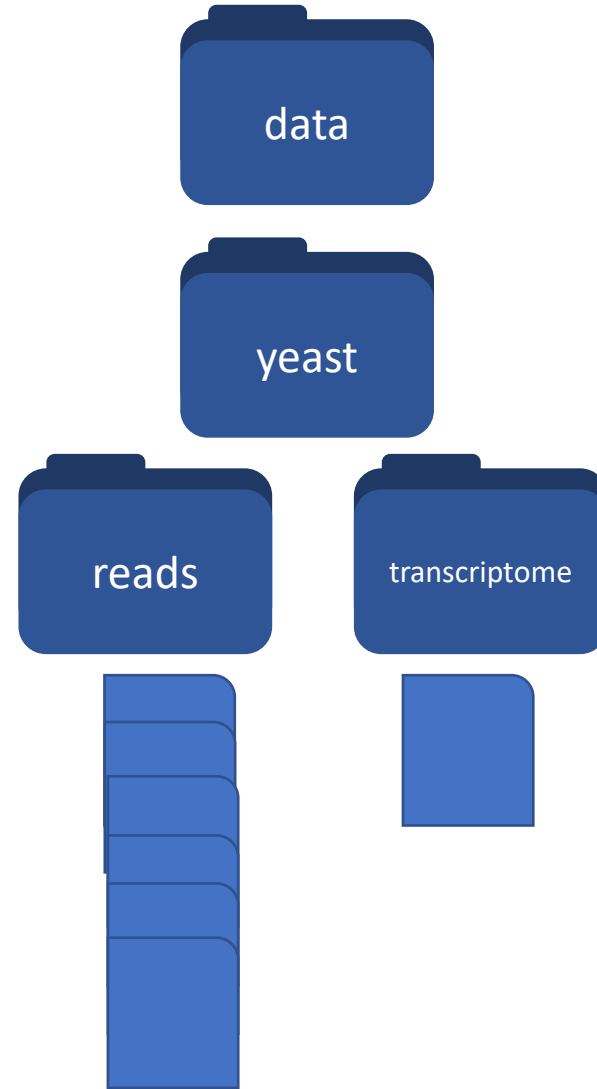
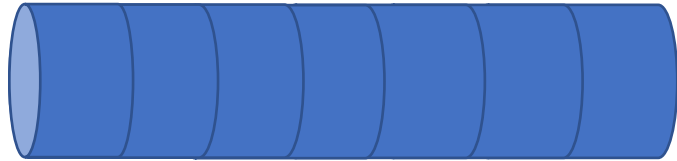
mix



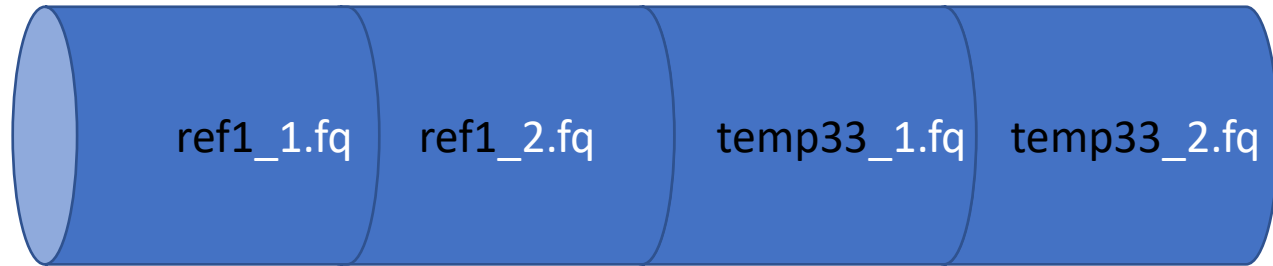
collect



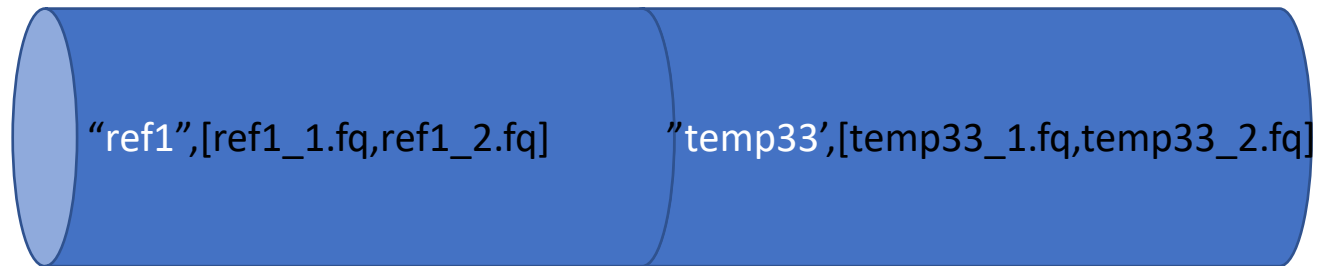
```
Channel.fromPath("/data/yeast/reads/*")
```



```
Channel.fromPath("/data/yeast/reads/*_{1,2}.fq")
```



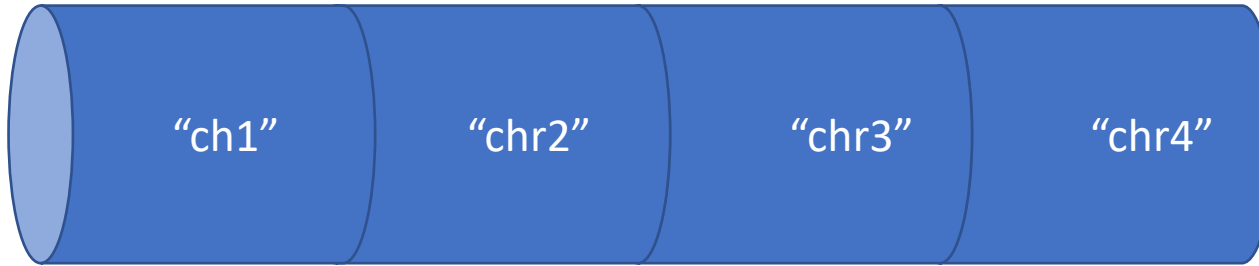
```
Channel.fromFilePairs("/data/yeast/reads/*_{1,2}.fq",size:2)
```



Filtering

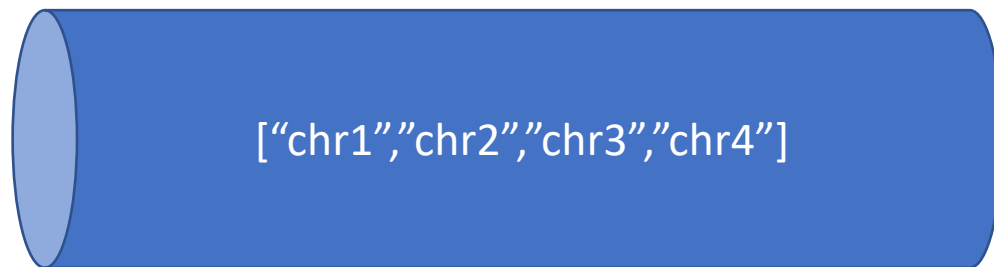


Transforming

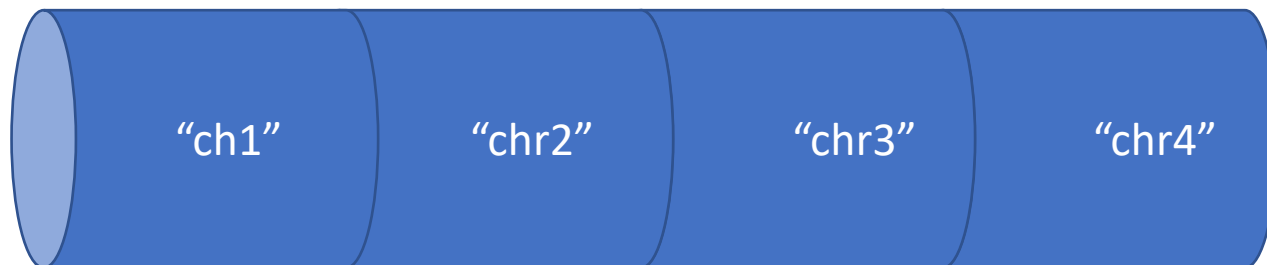


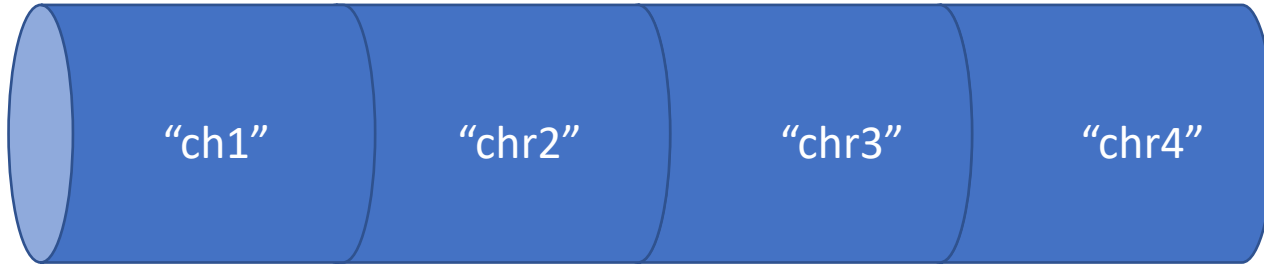
`map({it.replaceAll("chr","")})`



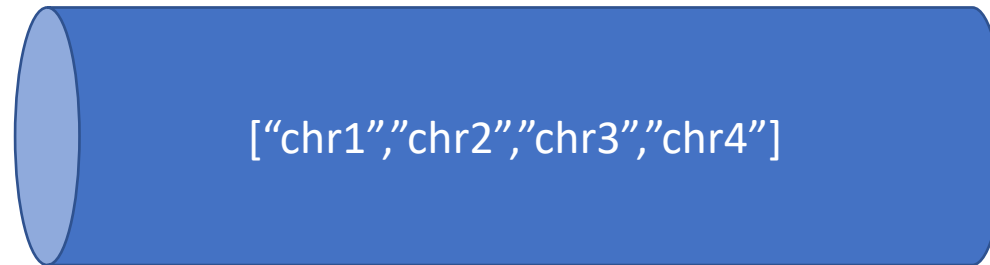


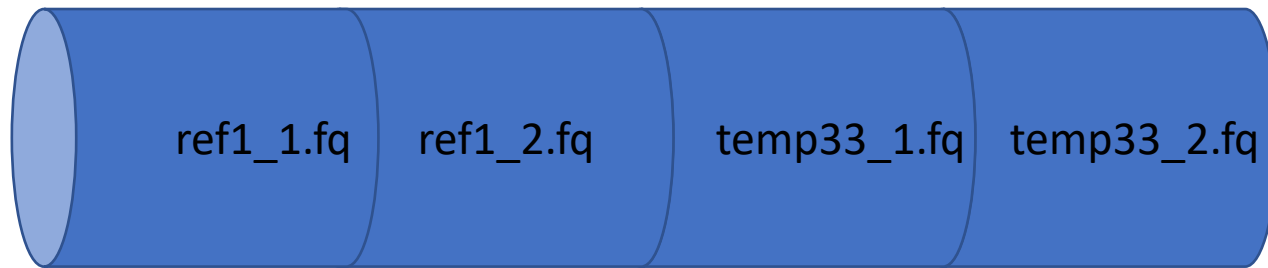
`flatten()`



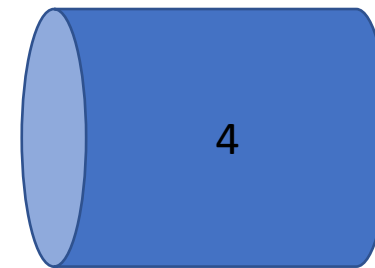


collect()





count





Availability

Nextflow Tower is an open-source monitoring and management platform for [Nextflow](#) workflows developed by [Seqera Labs](#).

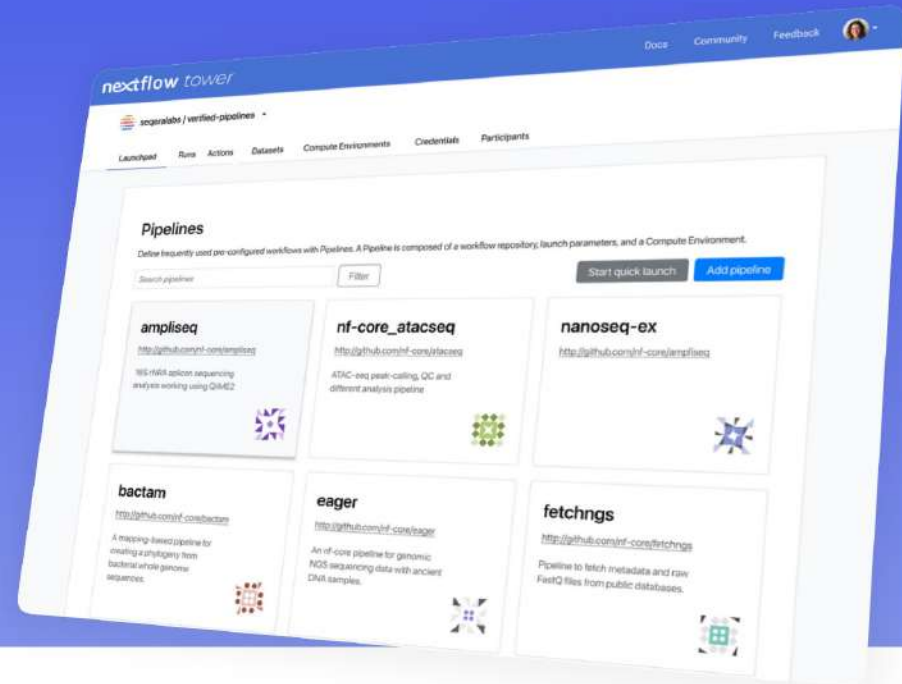
- The community version of Tower is available from <https://github.com/seqeralabs/nf-tower>. It can be deployed in a **user's own environment** and has features for **single users** to monitor their Nextflow pipelines, deployed anywhere.
- The fully-featured enterprise version of Tower is available from Seqera Labs. It can be deployed in a customers own **on-premise or cloud environment** and includes advanced workflow management, resource optimization and enterprise-grade support.

nextflow tower

Maximize productivity by enabling collaborative data analysis at scale, on-premises or in any cloud.

Try for free

Book a demo



Improve productivity

With a unified view of data, pipelines, results, and compute resources, users can collaborate more effectively and streamline analysis, data generation, and reporting.



Reduce costs

Optimise compute and storage costs, avoid expensive errors, and manage spending across projects and teams more effectively.



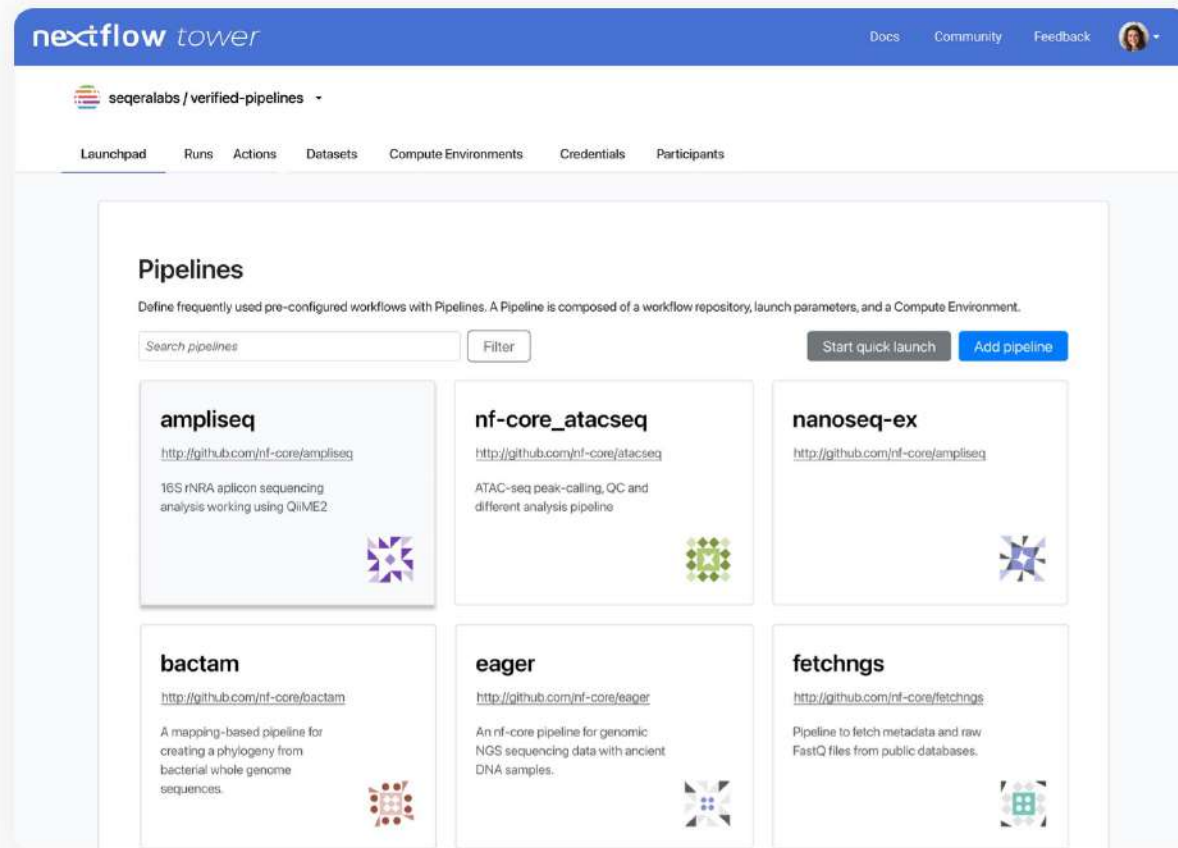
Remove complexity

Research teams can focus on the science that matters, and not on infrastructure engineering.



Simplify compliance

Ensure compliance and security with reliable, predictable, reproducible, and auditable pipeline execution.



Intuitive launchpad interface

Enable non-technical users to easily run pipelines via the intuitive Launchpad interface

nextflow tower

Docs Community Feedback

seqeralabs / verified-pipelines

Launchpad Runs Actions Datasets Compute Environments Credentials Participants

nf-core/rnaseq pipeline parameters

Upload params file

>_ Input/output options

Define where the pipeline should find input data and save output data.

input

?

Path to comma-separated file containing information about the samples in the experiment.

outdir

Path to the output directory where the results will be saved.

email

Email address for completion summary.

multiqc_title

MultiQC report title. Printed as page header, used for filename if not otherwise specified.

☒ **save_merged_fastq**

Save FastQ files after merging re-sequenced libraries in the results directory.

>_ Input/output options

- input
- outdir
- email
- multiqc_title
- save_merged_fastq

UMI options

Read filtering options

Show hidden params

Launch

Proven community pipelines

Easily access libraries of production-proven Nextflow community pipelines including nf-core among the others

nextflow tower

Docs Community Feedback









seqeralabs / verified-pipelines

Launchpad Runs Actions Datasets Compute Environments Credentials Participants

Runs

Search and manage workflow executions

Search workflow... ? Delete selected

<input type="checkbox"/> Workflow	User	Submit date	
<input type="checkbox"/>  nf-core/rnaseq ☆ id 2KoMK3bwMPMzHI •  silly_austin	suchita880	Nov 26	⋮
<input type="checkbox"/>  nf-core/rnaseq ☆ id tYtZLL4okxcP •  admiring_waddington	evanfioden	Nov 25	⋮
<input type="checkbox"/>  nf-core/ampliseq ☆ id jqMTeDWE2ru95 •  nauseous_panini	abhinav	Nov 25	⋮
<input type="checkbox"/>  nf-core/rnaseq ☆ id 5pDr25p9PqLsb •  intergalactic_brenner	evanfioden	Nov 24	⋮

Launch, manage, and monitor

Launch, manage, and monitor scalable, portable Nextflow pipelines from anywhere in real time



Compute Environments

[New Environment](#)

Compute Environments define the execution platform where a pipeline will run.

AWS_Batch_eu-west-1 primary

Status: AVAILABLE

[Clone](#)[Delete](#)

Id: 1voNN3Mve8r49EK6hFCN5v

Last activity: Nov 23, 2021, 5:42:27 PM

Slurm

Status: AVAILABLE

[Make primary](#)[Clone](#)[Delete](#)

Id: 2x9akyqwX6cnheXnnBBIMl

Last activity: Nov 23, 2021, 5:40:48 PM

Azure_Batch_east-us

Status: AVAILABLE

[Make primary](#)[Clone](#)[Delete](#)

Id: 7Qbwj1fIJQZksyaf34fnSj

Last activity: Nov 23, 2021, 3:30:08 PM

AWS_Batch_eu-west-3

Status: AVAILABLE

[Make primary](#)[Clone](#)[Delete](#)

Id: 97DcTqno5C8yYS71fhsw

Last activity: Nov 12, 2021, 5:26:17 PM

Your choice of cloud

- Run pipelines on any environment –
- from workstations to on-premises
- clusters to private, hybrid and public
- clouds

nextflow tower

Docs Community Feedback

segeralabs / verified-pipelines

Launchpad Runs Actions Datasets Compute Environments Credentials Participants

nf-core/rnaseq ☆
goofy_mandelbrot

Command line Parameters Configuration Execution log Reports Re-launch

MultiQC Reports

MultiQC
v1.10.1

General Stats
Biotype Counts
STAR_SALMON DESeq2 PCA plot
STAR_SALMON DESeq2 sample similarity
SALMON DESeq2 PCA plot
SALMON DESeq2 sample similarity
DupRadar
Picard

MultiQC

A modular tool to aggregate results from bioinformatics analyses across many samples into a single report.

This report has been generated by the [nf-core/maseq](#) analysis pipeline. For information about how to interpret these results, please see the [documentation](#).

Report generated on 2021-11-24, 17:05 based on data in: /tmp/nxf.VB2Hg086h9

General Statistics

Copy table Configure Columns 1 Plot Showing 11/11 rows and 24/30 columns.

Improve collaborative research

Collaborate and share data securely
among local and remote teams with rich
organization and workspace
management features

Compute platforms

Automatically provision, manage and scale compute environments in the cloud, or tap existing on-premises or cloud HPC and Kubernetes clusters for maximum flexibility.



Container Technologies

Leverage your choice of container technologies drawn from private or public registries ensuring reliable, reproducible, and seamless pipeline execution on any compute environment.



Source Code Management

Seamlessly integrate Nextflow Tower with your existing dev ops workflows, and access pipelines directly from your chosen source code management system or self-hosted Git servers.

GitHub 

 GitLab

 Bitbucket

Clients



Acknowledgement and further reading

1. <https://training.seqera.io/>
2. <https://carpentries-incubator.github.io/workflows-nextflow/01-getting-started-with-nextflow/index.html>
3. <https://seqera.io/tower/>

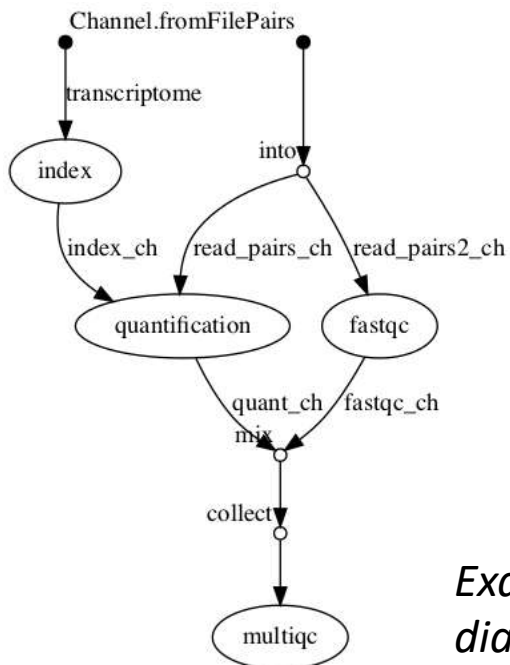
Opportunities

1. <https://nf-co.re/mentorships>

Exercises: Simple RNA-seq pipeline

We will follow a simple exercise that also captures what we have learned in the past few days. (Git, Conda, Docker)

Material: <https://github.com/ajodeh-juma/ngs-academy-africa-nfcore>



Example simple RNA-Seq pipeline in DAG format diagram from nf-core (<https://nf-co.re/sarek>)