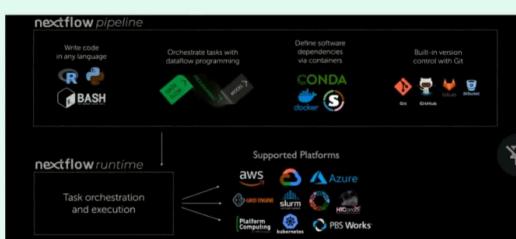




Nextflow: The Workflow Orchestrator

Nextflow: The Workflow Orchestrator



Nextflow is a powerful, open-source workflow management system designed for computational pipelines. It provides a domain-specific language (DSL) to define complex data analysis workflows.

- Parallelization: Efficiently executes tasks in parallel, maximizing computational resources.
- Reentrancy: Resumes interrupted workflows from the last completed step, saving time and resources.
- Modularity: Encourages breaking down workflows into reusable components for cleaner code.
- Portability: Runs seamlessly across local machines, HPC clusters, and cloud platforms.

Nextflow ensures that your bioinformatics pipelines are robust, efficient, and easily adaptable to different computational environments.

Modular Design with Nextflow DSL2



Clear, Maintainable Code

DSL2 promotes writing cleaner, more readable Nextflow code, making pipelines easier to understand and debug.



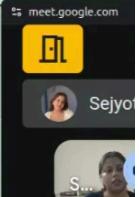
Reusable Modules

Develop and integrate reusable components (modules) for common bioinformatics tools and steps, accelerating pipeline development.



Continuous Integration

nf-core utilizes robust CI/CD practices to ensure pipeline stability, quality, and operation with eve



This modular approach enhances pipeline development efficiency and ensures long-term sustainability.

Why DSL matters in Nextflow?

- It defines how you write processes, workflows, channels, and parameters.
- Makes workflows:
 - Easy to reproduce
 - Portable across computing environments (e.g., local, cloud, HPC)
 - Modular and scalable

```
nextflow.enable.dsl=2

process sayHello {
    input:
        val name

    output:
        stdout

    script:
        """
        echo "Hello, $name!"
        """

    }
}
```

```
workflow {  
sayHello(name: 'Vibhanshu')  
}
```

Seamless Software & Compute Integration

Nextflow and nf-core pipelines are designed for maximum compatibility across diverse computational environments, ensuring your analyses run consistently wherever you need them.



Containerization

Full support for Singularity and Docker, packaging all dependencies for consistent execution.



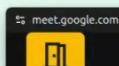
HPC Schedulers

Integrates with common High-Performance Computing (HPC) schedulers like SGE, PBS, LSF, and SLURM.



Cloud-Ready

Execute pipelines on major cloud platforms including Microsoft Azure, Amazon Web Services (AWS), and Google Cloud.



This extensive integration capability ensures your bioinformatics workflows are truly portable and scalable.

Launching Pipelines with Ease

nf-core provides intuitive tools to simplify the process of launching and running complex bioinformatics pipelines.

- **Web-Based Launch Wizard:** The [nf-co.re/launch](#) tool offers a user-friendly interface to configure pipeline parameters without needing to touch the command line.
- **Command-Line Efficiency:** For advanced users, powerful command-line tools enable quick setup, execution, and customization of pipelines.

Example: `nextflow run nf-core/rnaseq -params-file nf-params.json`

This command initiates the RNA-seq pipeline using parameters defined in a JSON file, showcasing the simplicity of execution.

Getting started with complex bioinformatics analyses has never been more straightforward.



Real-World Impact & Adoption

Driving Reproducible Research Forward

Nextflow and nf-core have been widely adopted across academic and industrial bioinformatics, revolutionizing how research is conducted and shared.

“ Academic Excellence ”

Numerous research institutions leverage nf-core pipelines for robust and shareable data analyses, from large-scale genomics to single-cell studies.

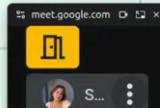
“ Industry Standard ”

Biotech and pharmaceutical companies integrate Nextflow workflows for reliable and scalable bioinformatics solutions in drug discovery and development.

“ Diverse Applications ”

Used for critical applications like RNA-seq analysis, viral genome sequencing, metagenomics, and much more, ensuring consistent and high-quality results.

The widespread adoption underscores their crucial role in accelerating scientific breakthroughs.



WHAT IS NEXTFLOW?

Before we start let's bake some cakes

- Ingredients
- Steps
- Order
- Dependencies
- Platform



1. Ingredients :

→ Analogous to input data, tools, and software components you need to run a workflow.

2. Steps :

→ Represents individual computational tasks or processes in the workflow.

3. Order :

→ Highlights that steps must execute in a particular sequence.

4. Dependencies :

→ Suggests that some steps depend on the results of others.

5. Platform :

→ Likely emphasizes Nextflow's ability to run on multiple computing environments (local, HPC clusters, cloud services like AWS, Google Cloud, etc.)

Simple Example of Nextflow Process:

```
process sayHello {  
    input:  
        val name  
  
        output:  
            stdout  
  
        script:  
            """  
            echo "Hello, $name!"  
            """  
  
}
```

The **four key components of a Nextflow process** are usually:

- 1. Process definition** → Defines the block of code as a process.
- 2. Input** → Data or channels coming into the process.
- 3. Output** → Results or channels produced by the process.
- 4. Script (or Command block)** → The actual commands or scripts executed by the process.

Channels

- Creating channels
- Sending data
- Receiving data
- Mixing channels
- Splitting channels
- Operators

- **Creating channels :**

→ How to create channels in Nextflow using commands like `Channel.fromPath`, `Channel.from`, etc.

- **Sending data :**

→ How processes send data into channels to be consumed by other processes.

- **Receiving data :**

→ How data flows from channels into processes as inputs.

- **Mixing channels :**

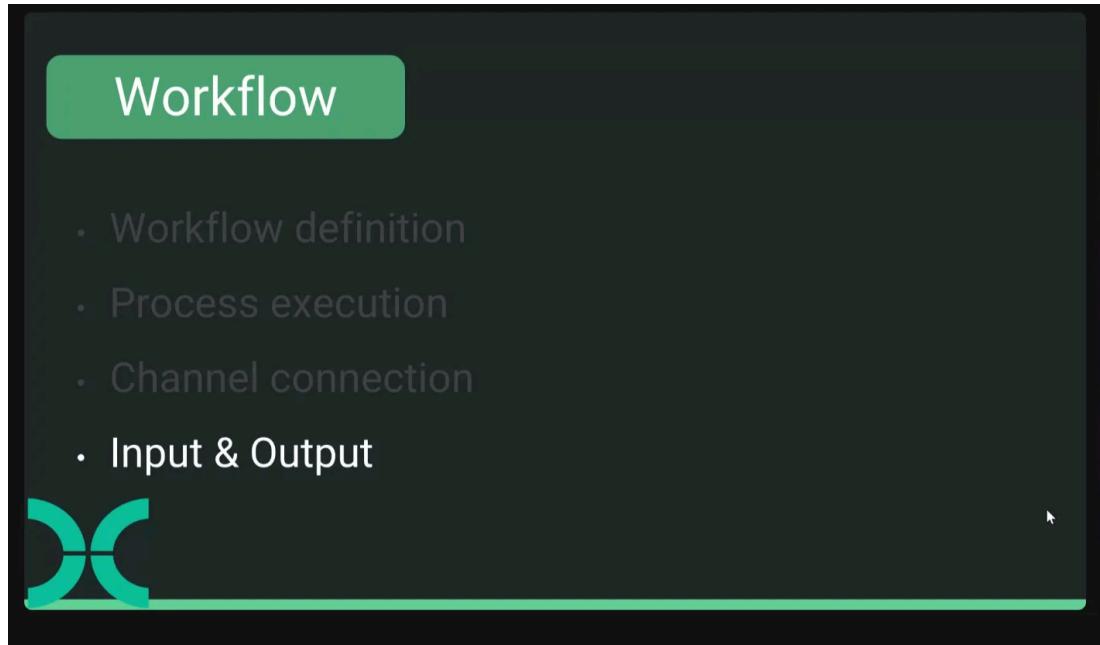
→ Combining multiple channels into one (e.g., `mix`, `combine`).

- **Splitting channels :**

→ Separating data from a channel into multiple outputs using operators.

- **Operators:**

→ Special methods provided by Nextflow to manipulate data in channels, such as `map`, `filter`, `flatMap`, `collect`, etc.



- **Workflow definition :**

→ How to define a workflow in Nextflow using the `workflow { ... }` block.

- **Process execution :**

→ How Nextflow runs the defined processes, managing parallelism and dependencies.

- **Channel connection :**

→ How data flows between processes via channels.

- **Input & Output :**

→ How workflows handle passing input data into processes and collecting outputs, connecting channels and processes to form the overall pipeline.

```
workflow {
    Channel.from('Alice', 'Bob', 'Charlie')
    .set { names }
```

```
sayHello(name: names)
```

```
}
```

EXAMPLE

```
#!/usr/bin/env nextflow  
  
process sayHello {  
    publishDir 'results', mode: 'copy'
```

```
    input:  
        val greeting
```

```
    output:  
        path 'output.txt'
```

```
    script:  
        """  
        echo '$greeting' > output.txt  
        """
```

```
}
```

```
params.greeting = 'Hola mundo!'  
  
workflow {  
    sayHello(params.greeting)  
}
```

Purpose:

- This script demonstrates a simple **Nextflow pipeline** that prints a greeting message into a file using **channels, process, and workflow structure**.

1 Shebang Line:

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

- Specifies that this script should be run using Nextflow.

2 Process Definition:

```

process sayHello {
    publishDir 'results', mode: 'copy'

    input:
        val greeting

    output:
        path 'output.txt'

    script:
        """
        echo '$greeting' > output.txt
        """
}

}

```

- **process sayHello** : Defines a process named `sayHello`.
- **publishDir 'results', mode: 'copy'**: Output will be copied to the `results/` directory.
- **input:** : Receives a value called `greeting`.
- **output:** : Produces a file `output.txt`.
- **script:** : Executes the command that echoes the greeting into `output.txt`.

3 Pipeline Parameters:

```
params.greeting = 'Hola mundo!'
```

- Sets a parameter `greeting` with the value `'Hola mundo!'`.

4 Workflow Block:

```

workflow {
    sayHello(params.greeting)
}

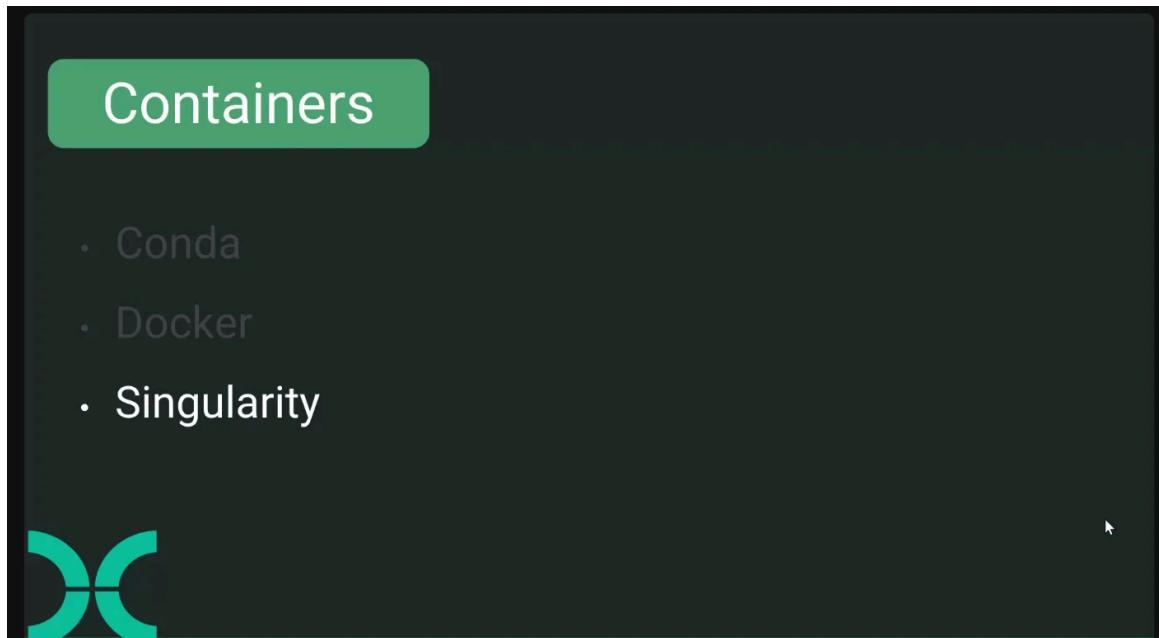
```

```
}
```

- Calls the process `sayHello`, passing the greeting parameter.

Overall Behavior of the Script:

1. Nextflow initializes the pipeline.
2. The `params.greeting` is passed as input to the `sayHello` process.
3. The process creates a file `output.txt` containing the text `'Hola mundo!'`.
4. The output file is saved in the `results/` directory.



Modules

- Tools
- Containers (Dependencies)
- Processes



Nextflow process definition written in the `main.nf` file, part of a pipeline module called `idxstats`. The goal of this process is to run `samtools idxstats` on input BAM files and produce index statistics files.

Here is a detailed explanation of the code:

✓ 1. Process Declaration

```
process SAMTOOLS_IDXSTATS {
```

Defines a Nextflow process named `SAMTOOLS_IDXSTATS`.

✓ 2. Metadata and Label

```
tag "$meta.id"  
label 'process_single'
```

- `tag "$meta.id"`: Adds a tag for easy logging and monitoring.
- `label 'process_single'`: Used for task scheduling (e.g., resource selection).

✓ 3. Container Setup

```
conda "bioconda::samtools=1.16.1"
container "${ workflow.containerEngine == 'singularity' && !task.ext.singularity_pull_docker_container ? 'https://depot.galaxyproject.org/singularity/samtools:1.16.1--h6899075_1' : 'quay.io/biocontainers/samtools:1.16.1--h6899075_1' }"
```

- Uses **Conda** and/or **Singularity/Docker container** to ensure reproducible environment.
- Automatically selects appropriate container source based on configuration.

✓ 4. Input Specification

```
input:
tuple val(meta), path(bam), path(bai)
```

- Inputs are:
 - `meta` : Metadata in tuple format.
 - `bam` : Path to the BAM file.
 - `: Path to the index file of the BAM.`

✓ 5. Output Specification

```
output:
tuple val(meta), path("*.idxstats"), emit: idxstats
path "versions.yml", emit: versions
```

- Outputs:
 - Index stats file (`.idxstats`).
 - Versions file (`versions.yml`) for reproducibility.

✓ 6. Execution Condition

when:

```
task.ext.when == null || task.ext.when
```

- Executes conditionally based on `task.ext.when` value (used for skipping or forcing execution).

✓ 7. Script Section

```
script:  
def args = task.ext.args ?: ""  
def prefix = task.ext.prefix ?: "${meta.id}"  
  
"""  
samtools idxstats \\  
--threads ${task.cpus - 1} \\  
$bam \\  
> ${prefix}.idxstats  
  
cat <<-END VERSIONS > versions.yml  
${task.process}  
END VERSIONS  
"""
```

- Builds a command to run `samtools idxstats` with CPU threading.
- Saves the result to `${prefix}.idxstats`.
- Saves process metadata to `versions.yml`.

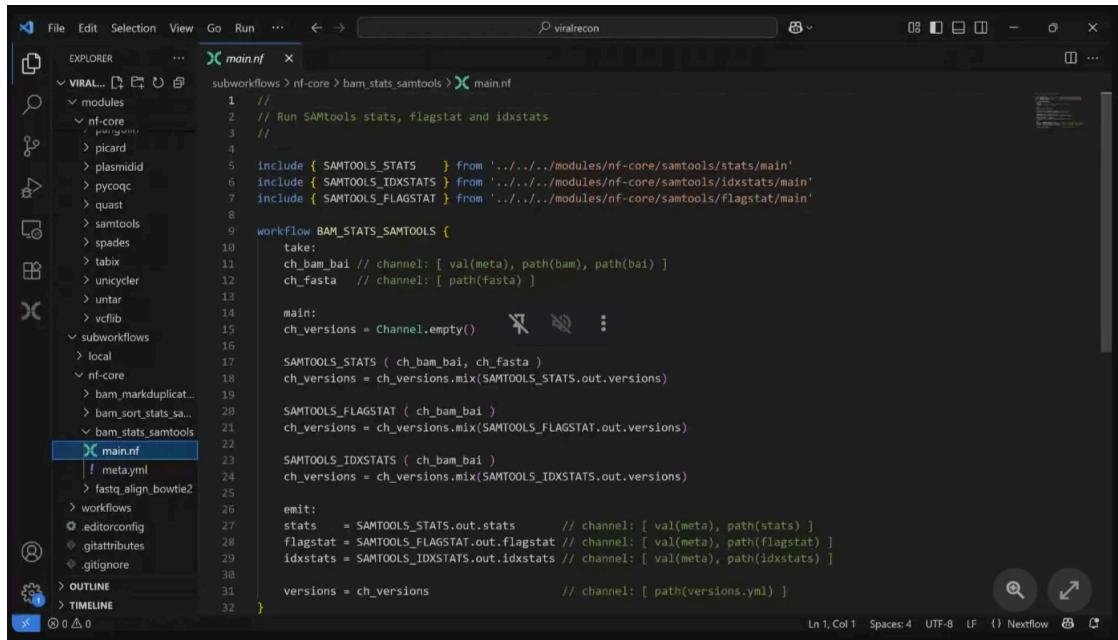
✓ 8. Purpose

This process is part of a larger Nextflow pipeline (likely viral sequence analysis based on folder structure).

Its role is to calculate alignment statistics for BAM files, which helps assess mapping quality, coverage, etc.

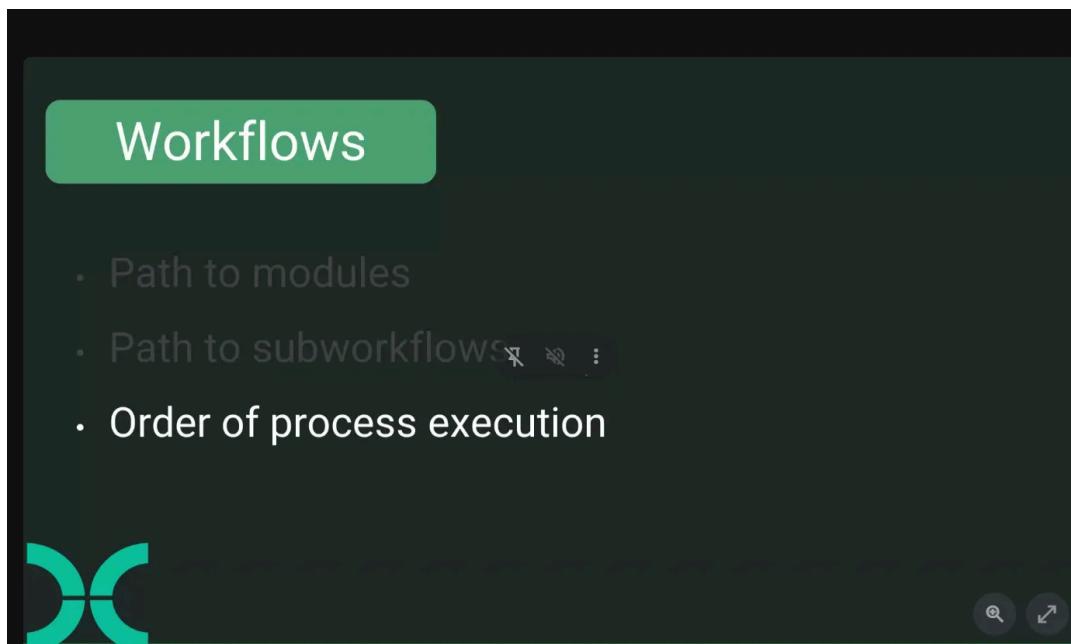
📁 Folder Context (from Explorer Sidebar)

- Located in: `modules/nf-core/samtools/idxstats/`
- Other related files:
 - `meta.yml`
 - `index`, `sort`, `stats`, etc.



The screenshot shows the Nextflow IDE interface. The main window displays the `main.nf` script, which includes code for running SAMtools stats, flagstat, and idxstats. The Explorer sidebar on the left shows the project structure under the `VIRAL...` root, including modules like nf-core, samtools, and subworkflows. The `main.nf` file is selected in the sidebar. The status bar at the bottom indicates the current line (Ln 1), column (Col 1), spaces (Spaces: 4), encoding (UTF-8), and other workflow details.

```
1 //  
2 // Run SAMtools stats, flagstat and idxstats  
3 //  
4  
5 include { SAMTOOLS_STATS } from '../../../../../modules/nf-core/samtools/stats/main'  
6 include { SAMTOOLS_IDXSTATS } from '../../../../../modules/nf-core/samtools/idxstats/main'  
7 include { SAMTOOLS_FLAGSTAT } from '../../../../../modules/nf-core/samtools/flagstat/main'  
8  
9 workflow BAM_STATS_SAMTOOLS {  
10     take:  
11         ch_bam_bai // channel: [ val(meta), path(bam), path(bai) ]  
12         ch_fasta // channel: [ path(fasta) ]  
13  
14     main:  
15         ch_versions = Channel.empty()  
16  
17         SAMTOOLS_STATS ( ch_bam_bai, ch_fasta )  
18         ch_versions = ch_versions.mix(SAMTOOLS_STATS.out.versions)  
19  
20         SAMTOOLS_FLAGSTAT ( ch_bam_bai )  
21         ch_versions = ch_versions.mix(SAMTOOLS_FLAGSTAT.out.versions)  
22  
23         SAMTOOLS_IDXSTATS ( ch_bam_bai )  
24         ch_versions = ch_versions.mix(SAMTOOLS_IDXSTATS.out.versions)  
25  
26         emit:  
27             stats    = SAMTOOLS_STATS.out.stats      // channel: [ val(meta), path(stats) ]  
28             flagstat = SAMTOOLS_FLAGSTAT.out.flagstat // channel: [ val(meta), path(flagstat) ]  
29             idxstats = SAMTOOLS_IDXSTATS.out.idxstats // channel: [ val(meta), path(idxstats) ]  
30  
31         versions = ch_versions                  // channel: [ path(versions.yml) ]  
32 }
```



A screenshot of the Visual Studio Code (VS Code) editor. The title bar shows the file name "main.nf". The editor displays the following Nextflow script:

```
main.nf
51 //----- WORKFLOW FOR VARIOUS
52 //----- WORKFLOW FOR VARIOUS
53 //----- WORKFLOW FOR VARIOUS
54 /*
55  */
56 if (params.platform == 'illumina') {
57 |   include ( ILLUMINA ) from './workflows/illumina'
58 } else if (params.platform == 'nanopore') {
59 |   include ( NANOPORE ) from './workflows/nanopore'
60 }
61 workflow NFCORE_VIRALRECON {
62   //
63   // WORKFLOW: Variant and de novo assembly analysis for Illumina data
64   //
65   if (params.platform == 'illumina') {
66     ILLUMINA ()
67   //
68   // WORKFLOW: Variant analysis for Nanopore data
69   //
70   } else if (params.platform == 'nanopore') {
71     NANOPORE ()
72   }
73 }
74 /*
75 |   RUN ALL WORKFLOWS
76 */
77 /*
78 */
79 /*
80 */
81 /*
82 */
83 */
```

A screenshot of a presentation slide from a platform like Miro or Notion. The slide has a dark background with a green header bar. In the top left corner, there are three small icons: a video camera, a document, and a person's profile. To the right of these icons, the text "Richard Agyekum (Presenting, annotating)" is displayed. The main title "Resources" is centered in a large, white, sans-serif font within a rounded green button-like shape. Below the title, there is a bulleted list of links:

- nf-core Github repository
- training.nextflow.io
- community.seqera.io
- nf-core slack channel

At the bottom left, there is a teal-colored logo consisting of two overlapping semi-circles. To its right is a yellow thumbs-up icon. Below the logo, the name "vibha kushwaha" is written in a small, white, sans-serif font.