

Introduction to Nextflow on the SCC

Instructors: Yun Shen & Andy Rampersaud
Research Computing Services – Information Services & Technology

Thursday October 1, 2026 (10:00am - 12:00pm)



Outline

- 1 What is Nextflow? Why use it?
- 2 Nextflow syntax & script structure
- 3 Running Nextflow on the SCC
- 4 Hands-On: Build & run a simple pipeline
- 5 Hands-On: Multi-step workflow & executors
- 6 nf-core pipelines + Live demos on SCC

Learning Objectives

By the end of this tutorial, you will be able to:

- 1 Understand Nextflow's core concepts**
Processes, workflows, channels, and how data flows between pipeline steps.
- 2 Read and write a basic Nextflow script**
Interpret .nf files, define processes, connect them in a workflow block.
- 3 Run a Nextflow pipeline on the SCC**
Use the local and SGE executors; configure resources via nextflow.config.
- 4 Adapt an existing pipeline to your data**
Modify parameters, swap inputs, and reuse published pipelines with minimal changes.
- 5 Find and run nf-core pipelines**
Browse nf-co.re, configure nf-core/rnaseq for BU SCC, and interpret results.

What is Nextflow?

A workflow system for creating scalable, portable, and reproducible workflows.



Designed for portability and reproducibility



Runs on your local machine, HPC schedulers like the SCC, or the cloud



Supports containers: Docker, Singularity, and Conda



Scripts are written in a Groovy-based DSL for defining workflows

Why use Nextflow?

A pipeline connects processing steps (“processes”) that transform input data into output results.



You have a collection of scripts → organize them into one modular workflow

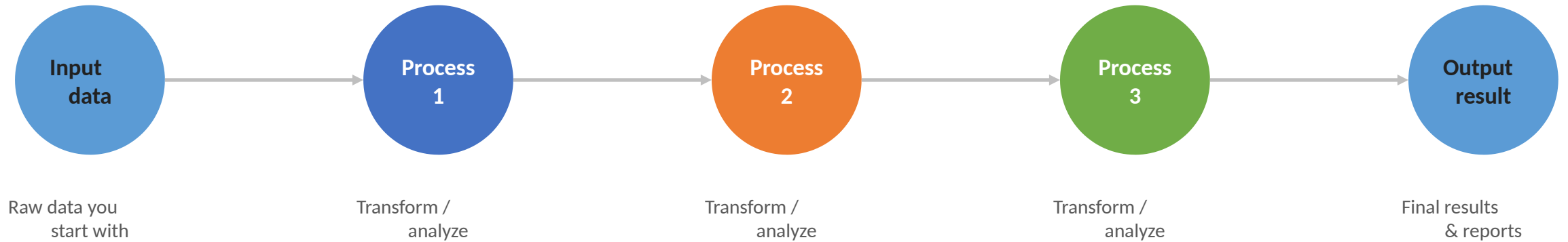


You inherited a Nextflow pipeline → reuse and adapt it for new data



You found a published pipeline on nf-core → run it on your data with minor config changes

Anatomy of a Pipeline



A pipeline (workflow) connects processing steps — each a Nextflow “process” — that transform input data into output results.

Basic Nextflow Script: hello-world.nf

```
/*  
 * Use echo to print 'Hello World!' to a file  
 */  
process sayHello {  
  
    output:  
    path 'output.txt'  
  
    script:  
    """  
    echo 'Hello World!' > output.txt  
    """  
}  
workflow {  
  
    main:  
    // emit a greeting  
    sayHello()  
}
```

A Nextflow script has two core components: one or more processes (what each step does) and the workflow block (how steps connect).

Using the SCC – Tutorial Accounts



Using tutorial accounts today – temporary accounts for use with tutorials



Refer to the table provided for your assigned account (e.g. tuta1 / password1)



Use of a tutorial account is optional if you already have an SCC account.

Log into the SCC – Open OnDemand

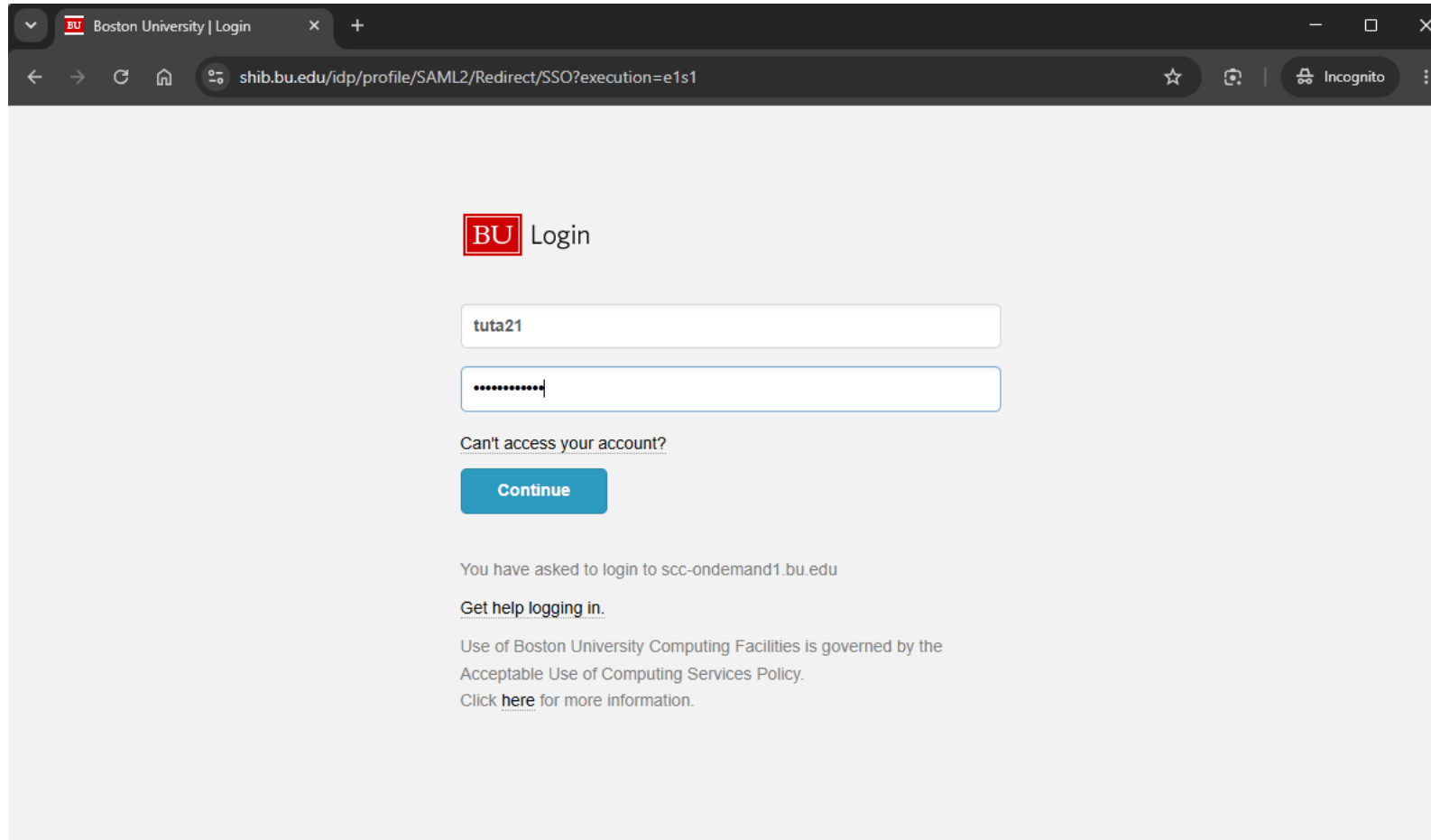


Tutorial accounts: scc-ondemand-tutorial.bu.edu



Normal SCC accounts: scc-ondemand.bu.edu (requires Duo two-factor)

Interactive Desktop on the SCC



The screenshot shows a web browser window with the title "Boston University | Login". The address bar displays the URL "shib.bu.edu/idp/profile/SAML2/Redirect/SSO?execution=e1s1". The page content includes the Boston University logo and the word "Login". There are two input fields: the first contains the username "tuta21", and the second contains masked characters ".....". Below the password field is a link "Can't access your account?". A blue "Continue" button is positioned below the link. Further down, the text "You have asked to login to scc-ondemand1.bu.edu" is displayed, followed by a link "Get help logging in.". At the bottom, a paragraph states: "Use of Boston University Computing Facilities is governed by the Acceptable Use of Computing Services Policy. Click [here](#) for more information."

Boston University | Login

shib.bu.edu/idp/profile/SAML2/Redirect/SSO?execution=e1s1

BU Login

tuta21

.....

[Can't access your account?](#)

Continue

You have asked to login to scc-ondemand1.bu.edu

[Get help logging in.](#)

Use of Boston University Computing Facilities is governed by the Acceptable Use of Computing Services Policy. Click [here](#) for more information.

Interactive Desktop on the SCC

The screenshot displays the SCC OnDemand web interface. The browser's address bar shows the URL `scc-ondemand1.bu.edu/pun/sys/dashboard/batch_connect/sessions`. The top navigation bar includes links for **SCC OnDemand**, **Files**, **Quotas**, **Login Nodes**, **Interactive Apps**, and **My Interactive Sessions**. The **Interactive Apps** dropdown menu is open, showing a list of applications: **Desktops** (with a sub-item **Desktop** circled in orange), **MATLAB**, **Mathematica**, **QGIS**, **SAS**, **STATA**, **Spyder**, and **VirtualGL Desktop**. Below this, the **Servers** section lists **Jupyter Notebook**, **MATLAB Server**, **RStudio Server**, **Shiny App Server**, **TensorBoard Server**, **VS Code Server**, and **Webserver**. The main content area features a **Research Center** section with a **Register here** link, a **Quota limit** warning (Using 882 GB of 900 GB), and a **Projectnb/tcwlabs-adsp** section with a 97% progress bar. A **Desktop (7490078)** session is highlighted in green, showing a **Host: scc-wal** and **Created at: 2026-09-08 03:32:36 EDT**. The **Time Remaining** is 23 hours and 53 minutes.

Interactive Desktop on the SCC

The screenshot shows the 'Desktop - SCC OnDemand' web interface. The browser address bar displays 'scc-ondemand1.bu.edu/pun/sys/dashboard/batch_connect/sys/desktop/session_contexts/new'. The page has a sidebar with 'Saved Settings' and 'Interactive Apps' lists. The main area is titled 'Desktop' and contains configuration fields for 'Use saved settings', 'List of modules to load', 'Working Directory', 'Initial command to run', 'Number of hours', 'Number of cores', 'Number of gpus', 'Project', 'Extra qsub options', and a 'Launch' button. Orange circles highlight the 'Number of hours' (12), 'Number of cores' (1), 'Project' (ysapp), and the 'Launch' button. A green oval highlights the 'Save settings and close' button.

Desktop - SCC OnDemand

scc-ondemand1.bu.edu/pun/sys/dashboard/batch_connect/sys/desktop/session_contexts/new

Home / My Interactive Sessions / Desktop

Saved Settings

- VS Code Server
 - vscode168
- Desktop
 - cascadelakeSession
 - desktop48
 - ysapp_desktop168
 - ysapp_gpu1
- RStudio Server
 - SigRepo

Saved Settings

- VS Code Server
 - vscode168
- Desktop
 - cascadelakeSession
 - desktop48
 - ysapp_desktop168
 - ysapp_gpu1
- RStudio Server
 - SigRepo

Interactive Apps

Desktops

- Desktop
 - MATLAB
 - Mathematica
 - QGIS
 - SAS
 - STATA
 - Spyder
 - VirtualGL Desktop
- Servers

Desktop

This app will launch an interactive desktop on a compute node.

Use saved settings
-- select saved settings --

List of modules to load (space separated)
[] [Select Modules]

Working Directory
[] [Select Directory]

The directory to start in. (Defaults to home directory.)

Initial command to run
xfce4-terminal

Number of hours
12

Number of cores
1

Number of gpus
[]

Project
ysapp

Extra qsub options
[]

☐ I would like to receive an email when the session starts

☐ Save settings

[] [Save settings and close]

Launch

* The Desktop session data for this session can be accessed under the data root directory.

Interactive Desktop on the SCC

The screenshot displays the 'My Interactive Sessions' page in the SCC interface. On the left, there are two panels: 'Saved Settings' and 'Interactive Apps'. The 'Saved Settings' panel lists various configurations like 'VS Code Server', 'vscode168', 'Desktop', 'cascadelakeSession', 'desktop48', 'ysapp_desktop168', 'ysapp_gpu1', 'RStudio Server', and 'SigRepo'. The 'Interactive Apps' panel lists 'Desktops' and various applications like 'MATLAB', 'Mathematica', 'QGIS', 'SAS', 'STATA', and 'Spyder'. The main area shows a session titled 'Desktop (444138)' with a status of '1 core' and 'Running'. A red 'Delete' button is visible. Below the session title, details are provided: Host (scc-wi2), Created at (2026-09-14 03:46:42 EDT), Time Remaining (11 hours and 59 minutes), Session ID (86c7f0f5-d076-4631-9a78-441911de38c2), List of modules to load (space separated), Working Directory, Number of hours (12), Number of cores (1), Number of gpus, GPU compute capability (6.0), Project (ysapp), and Extra qsub options. At the bottom, there are sliders for 'Compression' (1 (low) to 9 (high)) and 'Image Quality' (0 (low) to 9 (high)). A blue 'Connect to Desktop' button is highlighted with an orange circle, and a 'View Only (Share-able Link)' button is also present.

Home / My Interactive Sessions

Saved Settings

- VS Code Server
- vscode168
- Desktop
- cascadelakeSession
- desktop48
- ysapp_desktop168
- ysapp_gpu1
- RStudio Server
- SigRepo

Interactive Apps

- Desktops
- Desktop
- MATLAB
- Mathematica
- QGIS
- SAS
- STATA
- Spyder

Desktop (444138) 1 core Running

Host: scc-wi2 [Delete](#)

Created at: 2026-09-14 03:46:42 EDT

Time Remaining: 11 hours and 59 minutes

Session ID: 86c7f0f5-d076-4631-9a78-441911de38c2

List of modules to load (space separated):

Working Directory:

Number of hours: 12

Number of cores: 1

Number of gpus:

GPU compute capability: 6.0

Project: ysapp

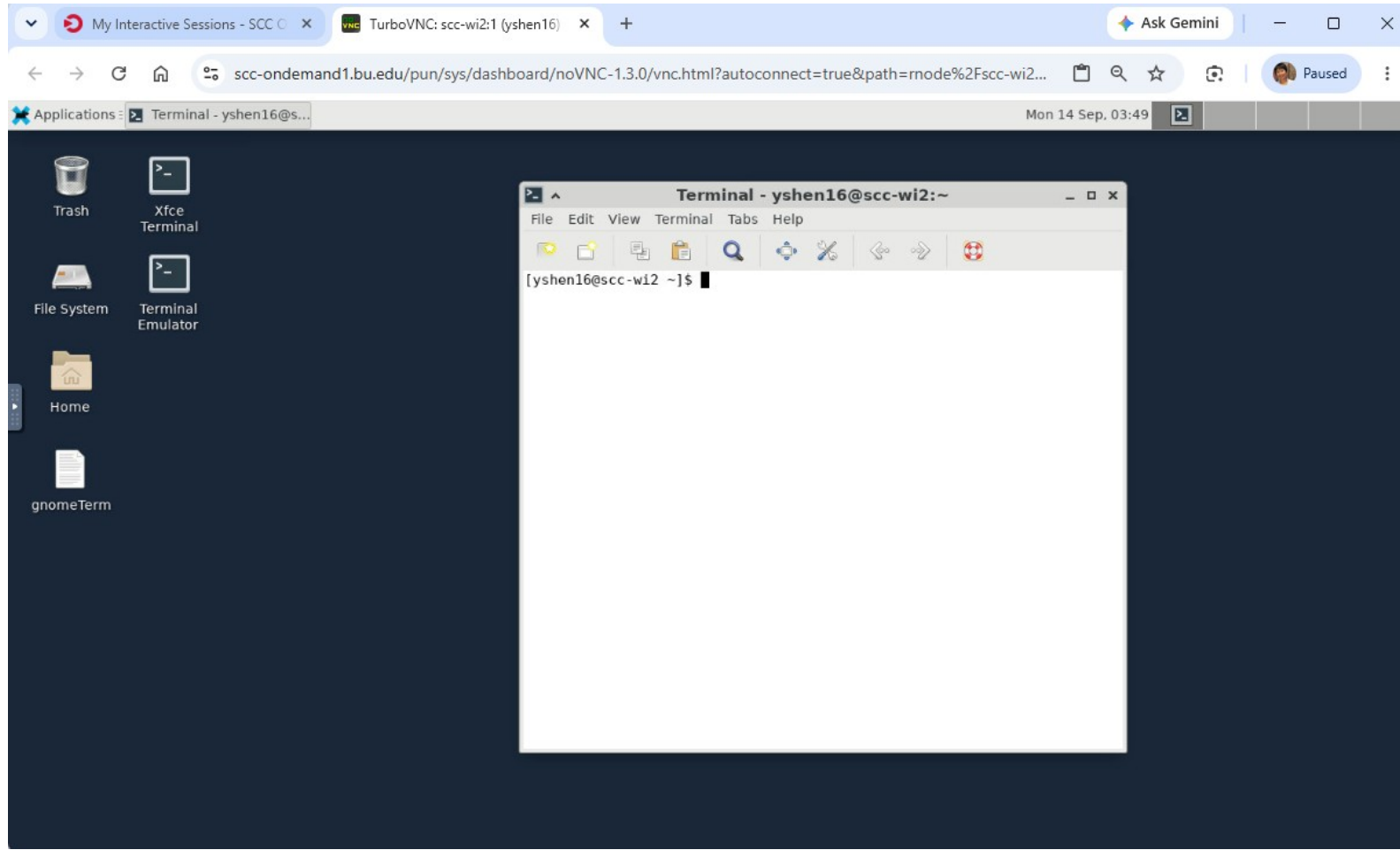
Extra qsub options:

Compression: 1 (low) to 9 (high)

Image Quality: 0 (low) to 9 (high)

[Connect to Desktop](#) [View Only \(Share-able Link\)](#)

Interactive Desktop on the SCC



Hands-On Activity – Prepare

```
# 1. Download tutorial materials
wget \
https://scv.bu.edu/examples/bioinformatics/nextflow/tutorial/IntroNextflowOnSCC.tar.gz
mkdir nf_tutorial
tar -xzf IntroNextflowOnSCC.tar.gz -C nf_tutorial/
cd nf_tutorial/

# 2. Set up environment (Singularity cache + load Nextflow)
source setup.sh
```

First we need to get the tutorial materials which will be demoed in the following steps

Hands-On Activity – Warmup

```
$ echo 'Hello World!'  
Hello World!
```

```
$ echo 'Hello World!' > output.txt
```

```
$ cat output.txt  
Hello World!
```

Run these directly in the terminal before touching Nextflow – same idea as the pipeline we're about to run, just without a workflow around it.

Hands-On Activity – Run the Pipeline

The Nextflow starter script is already copied to your working directory – review it, then launch and watch it execute.

```
$ module load nextflow/25.04.7
```

```
$ nextflow run hello-world.nf
```

```
N E X T F L O W ~ version 25.x.x  
Launching `hello-world.nf` [big_hilbert] DSL2 - revision:  
1a2b3c4d5e  
executor > local (1)  
[a1/b2c3d4] process > sayHello [100%] 1 of 1 ✓
```

Hands-On Activity – Find the Output & Logs

Nextflow creates a work directory (e.g. work/a1/b2c3d4...) for each process run.



.command.sh – the exact command Nextflow ran (most useful for troubleshooting)



.command.out / .command.err – stdout / stderr from the process



.command.log – complete combined log output



.exitcode – the exit code of the command

Hands-On: Multi-step Workflow

multi-steps.nf:

```
params.name = "World"

process GREET {
  input:  val name
  output: path 'greeting.txt'
  script:
  """
  echo "Hello, ${name}!" > greeting.txt
  """
}

process SHOUT {
  input:  path greeting
  output: path 'greeting_upper.txt'
  script:
  """
  tr '[:lower:]' '[:upper:]' < ${greeting} \
    > greeting_upper.txt
  """
}

process REPORT {
  publishDir "results/", mode: 'copy'
  input:  path upper
  output: path 'report.txt'
  script:
  """
  echo "=== Report ===" > report.txt
  cat ${upper}           >> report.txt
  """
}

workflow {
  name_ch = Channel.of(params.name)
  GREET(name_ch)
  SHOUT(GREET.out)
  REPORT(SHOUT.out)
}
```

Hands-On: Multi-step Workflow

Code explanations:

1. Three chained processes:
GREET → SHOUT → REPORT
2. input/output channels:
Demonstrates channels, publishDir, and data flow between steps.
3. Final output written to
results/report.txt

Commands to run:

```
$ module load nextflow/25.04.7  
$ nextflow run multi-step.nf \  
  --name "$USER"
```

Expected output (results/report.txt):

```
=== Pipeline Report ===  
Greeting (uppercase):  
HELLO, YSHEN16! WELCOME TO THE SCC.  
Word count: 6
```

Running Nextflow on the SCC: Executors

Nextflow supports various execution backends (called 'Executors'). Choose the right one for your workload.

	local	sge
What it does	Runs each task directly on the node where Nextflow is launched	Submits each task as a separate batch job to the SGE cluster queue
Best for	Quick tests, debugging, small datasets	Production runs, large datasets, computationally intensive workflows
Resources	Limited to the current node's CPUs and RAM	Full cluster — tasks run in parallel across many nodes simultaneously
How to activate	Default — no configuration needed	Add to nextflow.config: process { executor = 'sge' } Or: -profile sge on the command line
Typical use	Development & validation before scaling up	Real analysis workloads (align, assemble, call variants, ...)

Switching Executors



Start local, scale with SGE

Always develop and test your pipeline with the local executor first — it's faster to iterate. Once it works, switch to SGE to run on real data.



One line to switch

Add `process { executor = 'sge' }` to `nextflow.config`, or pass `'-profile sge'` on the command line. No changes to your `.nf` script are needed.



Set your SCC project

SGE requires a project allocation. Set it in `nextflow.config`:
`clusterOptions = '-P your_project -l h_rt=01:00:00'`, for flexibility, use a 'project' parameter, and pass the project name at runtime



Nextflow handles job submission

Each process becomes an independent SGE job. Nextflow monitors them, retries failures, and wires outputs between jobs automatically.

Configuring the SGE Executor

```
// nextflow.config

profiles {

  sge {
    process {
      executor      = 'sge'
      clusterOptions = '-P your_project \
                        -l h_rt=00:30:00'

      cpus          = 1
      memory        = '4 GB'
    }
  }
}
```

Place nextflow.config in the same directory as your .nf file. Replace 'your_project' with your SCC project name (e.g. tutorial). Use '-profile sge' on the command line to activate.

Using nf-core Pipelines on the SCC

```
# 1. Clone the nf-core/rnaseq pipeline
# Note: Clone the specific version the tutorial has tested.
# the default has new updates will break the pipeline
git clone --branch 3.26.0 \
  https://github.com/nf-core/rnaseq.git pipeline/rnaseq

# 2. Copy the customized SCC config into the pipeline
cp scc_sge.config pipeline/rnaseq/conf/

# 3. Quick test run (built-in small dataset)
nextflow run pipeline/rnaseq/main.nf \
  -profile singularity,test \
  --outdir results_test/
```

nf-core hosts 139+ curated Nextflow pipelines (nf-co.re/pipelines). Clone the one you are interested in to your project space for full control. Always test with '-profile test' before running on real data.

nf-core/rnaseq: Pipeline Overview

Version 3.26.0 · Full RNA-seq: FASTQ → expression matrices + QC report

- ✓ **Input**
Paired or single-end FASTQ files (or pre-aligned BAMs) declared in a CSV samplesheet
- ✓ **Alignment**
STAR (default), HISAT2, or skip alignment and use Salmon/Kallisto pseudo-alignment
- ✓ **Quantification**
Salmon, RSEM, featureCounts, StringTie, Kallisto — flexible per experiment needs
- ✓ **QC**
FastQC, RSeQC, Qualimap, dupRadar, DESeq2 clustering — aggregated into one MultiQC report
- **Containers**
Every tool ships as a Singularity/Docker image — no software installation required
- **Portability**
One command runs locally, on SGE/SLURM clusters, or in the cloud — same pipeline, same results

nf-core/rnaseq: Pipeline Stages

Nine sequential stages — from raw reads to expression matrices and QC report.

#	Stage	Key Tools
1	Read QC & Trimming	FastQC · fastp / Trim Galore!
2	UMI Extraction (optional)	UMI-tools · UMICollapse
3	rRNA / Contamination Removal (optional)	SortMeRNA · RiboDetector · BBSplit
4	Alignment	STAR (default) · HISAT2 · Bowtie2
5	BAM Processing	SAMtools sort/index · Picard MarkDuplicates
6	Quantification	Salmon · RSEM · featureCounts · StringTie · Kallisto
7	Coverage Tracks	BEDTools genomecov · UCSC bedGraphToBigWig
8	BAM Quality Control	RSeQC · Qualimap · dupRadar · Preseq
9	Reporting	DESeq2 QC · MultiQC

nf-core/rnaseq: Repository Structure

rnaseq/

```
├── main.nf                # Entry point
├── nextflow.config        # Parameters, profiles, resource limits
├── nextflow_schema.json   # Parameter validation schema
├── conf/
│   ├── base.config        # Default resource tiers (process_low/medium/high)
│   ├── scc_sge.config      # BU SCC SGE configuration ← our custom config
│   ├── test.config        # Built-in small test dataset
│   ├── rapid_quant.config  # Salmon-only fast mode (skip alignment)
│   └── modules/           # Per-tool parameter overrides (27 files)
├── workflows/rnaseq/main.nf # Core workflow logic (all 9 stages)
├── subworkflows/
│   ├── local/             # Pipeline-specific subworkflows (5)
│   └── nf-core/           # Reusable nf-core subworkflows (20+)
├── modules/
│   ├── local/             # Custom modules (3: DESeq2 QC, STAR upgrade, ...)
│   └── nf-core/           # Tool wrappers (77: STAR, Salmon, RSeQC, ...)
├── assets/
│   └── samplesheet.csv     # Example samplesheet (paired + single-end)
```

Key Configuration Parameters

All parameters can be set in nextflow.config (params block) or passed on the command line as --param value.

Parameter	Default	Description
--genome	—	iGenomes genome ID (e.g. GRCh38, mm10)
--fasta / --gtf	—	Explicit genome FASTA and GTF paths (alternative to --genome)
--save_reference	false	Save built indexes for reuse on subsequent runs
--aligner	star_salmon	Aligner: star_salmon · star_rsem · hisat2
--skip_alignment	false	Skip alignment — use Salmon/Kallisto pseudo-alignment only
--pseudo_aligner	salmon	Pseudo-aligner: salmon · kallisto (used alongside or instead of aligner)
--trimmer	trimgalore	Trimmer: trimgalore · fastp
--skip_trimming	false	Skip adapter/quality trimming
--min_trimmed_reads	10 000	Discard samples with fewer reads after trimming
--remove_ribo_rna	false	Enable rRNA removal (tools: sortmerna · bowtie2 · ribodetector)
--skip_qc	false	Skip all post-alignment QC steps
--skip_multiqc	false	Skip final MultiQC aggregation report

Configuring nf-core/rnaseq for BU SCC

```
// conf/scc_sge.config - BU SCC SGE configuration

params.project = null    // pass as: --project tutorial

process {
    executor = 'sge'
    penv      = 'omp'      // emit #$ -pe omp N (SCC requires PE, not -l slots=N)

    // clusterOptions closure: auto-computes mem_per_core and h_rt
    // from each task's cpus/memory/time - no per-process edits needed.
    clusterOptions = {
        def mem_per_core = (int) Math.max(1,
            Math.ceil(task.memory.toGiga() / task.cpus))
        def total_sec = (long) task.time.toSeconds()
        def h_rt = String.format('%02d:%02d:%02d',
            (int)(total_sec / 3600),
            (int)((total_sec % 3600) / 60),
            (int)(total_sec % 60))
        "-P ${params.project} -l h_rt=${h_rt} -l mem_per_core=${mem_per_core}G -V"
    }

    withLabel: process_medium { cpus = 6; memory = '36 GB'; time = '8 h' }
    withLabel: process_high   { cpus = 12; memory = '72 GB'; time = '16 h' }
}

singularity.enabled = true
singularity.cacheDir = "${launchDir}/.singularity_cache" // ← cwd
```

Running nf-core/rnaseq on BU SCC

```
# 1. Set Singularity image cache (current working directory)
mkdir -p .singularity_cache
export NXF_SINGULARITY_CACHEDIR=$PWD/.singularity_cache

# 2. Load Nextflow
module load nextflow/25.04.7

# 3. Run the pipeline
nextflow run pipeline/rnaseq/main.nf \
  -profile singularity \
  -c pipeline/rnaseq/conf/scc_sge.config \
  --project YOUR_PROJECT \
  --input samplesheet.csv \
  --outdir results/ \
  --genome GRCh38 \
  --aligner star_salmon
```

-profile singularity	use Singularity containers (system-wide on SCC, no module needed)
-c scc_sge.config	activate SGE executor + BU SCC resource settings
--project YOUR_PROJECT	SCC project name (sets -P flag and Singularity cache path)
--genome GRCh38	iGenomes reference (or use --fasta / --gtf for custom genome)
--aligner star_salmon	STAR alignment → Salmon quantification (pipeline default)

Demo 1: Rapid Quantification — Overview

Goal: quantify gene expression for 5 RNA-seq samples as fast as possible — no alignment step, using Salmon pseudo-alignment.



What it uses

Built-in nf-core test dataset (small synthetic FASTQ files downloaded automatically from GitHub). No local data files needed.



Profiles activated

singularity — containers pulled from quay.io/biocontainers/ (no software install needed)
test — loads built-in small test samplesheet + reference genome
rapid_quant — skips STAR alignment and all BAM-dependent QC steps



What runs

FastQC · Trim Galore! · BBSplit · Salmon pseudo-alignment · per-sample MultiQC



What is skipped

STAR genome index · STAR alignment · BAM processing · BigWig · RSeQC · Qualimap · DESeq2 QC



Expected outcome

Completes in ~3-5 min on the local node. Per-sample salmon/quant.sf files + merged TPM matrix + per-sample MultiQC reports.

Demo 1: Config & Run Command

conf/rapid_quant.config (key settings highlighted in green)

```
// conf/rapid_quant.config

params {
  skip_alignment = true      // ← no STAR, fastest path
  pseudo_aligner = 'salmon' // ← Salmon pseudo-alignment
  skip_quantification_merge = true
  skip_multiqc = false
  skip_bigwig = true        // ← no coverage tracks
  skip_stringtie = true
  skip_markduplicates = true
  skip_biotype_qc = true
  skip_dupradar = true
  skip_qualimap = true
  skip_rseqc = true
  skip_preseq = true
  skip_deseq2_qc = true
}
```

Run command

```
# Set Singularity cache to current working directory
export NXF_SINGULARITY_CACHEDIR=$PWD/.singularity_cache

# Load Nextflow
module load nextflow/25.04.7

# Run Demo 1 - rapid quantification with built-in test data
nextflow run pipeline/rnaseq/main.nf \
  -profile singularity,test,rapid_quant \
  --outdir results/demo1_rapid/
```

What Nextflow does:

- 1 Reads samplesheet from GitHub test URL
- 2 Pulls Singularity images from quay.io (cached after first run)
- 3 Runs FastQC → Trim Galore! → BBSplit → Salmon
- 4 Generates per-sample MultiQC + merged count matrix

Demo 1: Output Files & What They Mean

Output structure (green = most important files)

```
results/demo1_rapid/
├── {sample}/
│   ├── salmon/quant.sf          # transcript TPM + raw counts ← key output
│   ├── fastqc/                 # raw + trimmed read QC
│   ├── trimgalore/             # trimming reports
│   ├── multiqc/                # per-sample MultiQC report ← open this
│   └── bbsplit/                # contaminant filter stats
├── salmon/
│   ├── salmon.merged.gene_counts.tsv
│   └── salmon.merged.gene_tpm.tsv # merged TPM matrix ← key output
└── pipeline_info/
    ├── execution_report.html    # CPU/memory timeline
    └── execution_trace.txt      # per-task resource use
```

Key files to check

File	What it contains
quant.sf	Salmon transcript-level TPM + raw counts per sample
salmon.merged.gene_tpm.tsv	Gene-level TPM matrix — all samples in one table
multiqc_report.html	Interactive QC: read quality, trimming, mapping rate
execution_report.html	Pipeline timeline — CPU, memory, duration per task
execution_trace.txt	Tab-separated: exit status, realtime, %cpu, rss per task

Demo 2: Full Pipeline — Overview

Goal: run the complete nf-core/rnaseq pipeline with STAR alignment, BAM QC, BigWig tracks, and DESeq2 — then switch to SGE with -resume.



What it adds over Demo 1

STAR genome index · STAR spliced alignment → BAM · Picard MarkDuplicates · RSeQC · Qualimap · dupRadar · BigWig coverage tracks · DESeq2 PCA & clustering · aggregated MultiQC



Profiles activated

singularity — pull containers automatically
test — same built-in test data as Demo 1 (no local data needed)



Demo 2a: local executor

Runs all 227 processes on the current node. Completed in ~6 min with test data.



Demo 2b: switch to SGE with -resume

Add `-c scc_sge.config --project YOUR_PROJECT` — Nextflow resubmits only un-cached tasks as SGE batch jobs. Completed tasks from 2a are reused instantly.



Expected outcome

Full BAM files, stranded BigWig tracks, gene count matrix, one aggregated multiqc_report.html covering all 5 samples.

Demo 2: Config & Run Commands

conf/scc_sge.config (green = SGE settings, orange = clusterOptions
close

```
// conf/scc_sge.config

params.project = null    // pass --project YOUR_PROJECT

process { executor = 'sge' // ← submit each task as SGE job
          penv = 'omp'     // ← required by SCC SGE (not -l slots=N)

// Closure: auto-computes mem_per_core and h_rt
// from each task's cpus / memory / time — no hardcoding
clusterOptions = {
  def mem_per_core = (int) Math.max(1,
    Math.ceil(task.memory.toGiga() / task.cpus))
  def h_rt = String.format('%02d:%02d:%02d', ...)
  "-P ${params.project} -l h_rt=${h_rt} -l mem_per_core=${mem_per_core}G -V"
}
withLabel:process_medium { cpus=6; memory='36 GB'; time='8h' }
withLabel:process_high   { cpus=12; memory='72 GB'; time='16h' }
}

singularity.enabled = true
singularity.cacheDir = "${launchDir}/.singularity_cache" // ← cwd
```

Run commands

```
# Demo 2a - Full pipeline, local executor (for testing)
nextflow run pipeline/rnaseq/main.nf \
  -profile singularity,test \
  --outdir results/demo2_star/

# Demo 2b - Same pipeline, submit tasks to SGE cluster
nextflow run pipeline/rnaseq/main.nf \
  -profile singularity,test \
  -c pipeline/rnaseq/conf/scc_sge.config \
  --project YOUR_PROJECT \
  --outdir results/demo2_sge/ \
  --resume                # reuse cached tasks from 2a
```

Why -resume matters:

Nextflow caches each completed task by its inputs + code hash. Adding -c scc_sge.config resubmits only tasks that haven't run yet — so switching from local to SGE reuses everything already done.

Practically: demo 2a runs locally to verify, then 2b scales the same run to the cluster.

Demo 2: Output Files & What They Mean

Output structure (green = key files to open first)

results/demo2_star/

```
├── star_salmon/
│   ├── {sample}/*.bam      # aligned reads + .bai index ← alignments
│   ├── bigwig/             # .bw coverage tracks (fwd, rev, combined)
│   ├── featurecounts/      # gene-level count matrix ← counts
│   ├── rseqc/              # RSeqC: read distribution, junction plots
│   ├── qualimap/           # Qualimap: BAM QC metrics
│   ├── dupradar/           # PCR duplication assessment
│   ├── stringtie/          # transcript assemblies (.gtf)
│   └── deseq2_qc/          # PCA + clustering heatmap ← QC plots
├── salmon/
│   ├── salmon.merged.gene_tpm.tsv
│   └── deseq2_qc/          # DESeq2 QC on Salmon counts
├── multiqc/star_salmon/    # aggregated MultiQC report ← open this
└── pipeline_info/         # timeline, trace, DAG
```

Key files to check

File / Dir	What it contains
{sample}/*.bam	Aligned reads — load in IGV to inspect
bigwig/*.bw	Coverage tracks — strand-specific; load in IGV
featurecounts/*.txt	Gene-level count matrix (all samples)
deseq2_qc/	PCA plot + sample-distance heatmap — assess batch effects
multiqc/star_salmon/multiqc_report.html	Aggregated QC for all samples and tools — start here
execution_report.html	Per-task CPU, memory, walltime — tune resources

Nextflow Resources

Key links for learning, support, and finding pipelines:



Nextflow Documentation

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

Full language reference, executor docs, config options



nf-core Documentation

<https://nf-co.re/docs/>

Pipeline usage, contribution guides, nf-core tools



Nextflow Training (self-paced)

<https://training.nextflow.io/latest/>

This tutorial follows the "Hello Nextflow" section



nf-core Pipelines

<https://nf-co.re/pipelines/>

139+ curated, peer-reviewed community pipelines



Nextflow Slack workspace

<https://www.nextflow.io/slack-invite.html>

Community support, announcements, Q&A



nf-core Slack workspace

<https://nf-co.re/join/slack/>

Pipeline-specific channels, community help

Questions

All Nextflow and SCC questions welcome. For those we can't answer now; we can get back you with an answer for later.



Yun Shen – yshen16@bu.edu



Andy Rampersaud – aramp10@bu.edu



General SCC help – email: help@scc.bu.edu

web: <http://www.bu.edu/tech/support/research/system-usage/support-request/>

Tutorial Survey



Navigate to rcs.bu.edu/eval – select the correct tutorial under “Tutorial you attended”



Tutorial feedback survey – thanks for coming!

Thank you!!!