Reporting and config files

Pre-hackathon training — Kemri-Wellcome Trust

March 12th 2025
Cris Tuñí-i-Domínguez
c.tuni.dominguez@gmail.com

A bit about me

Cris Tuñí i Domínguez

ctuni.dev

- PhD in biomedicine student
- Bioinformatics scientist at Flomics Biotech S.L.
- From Barcelona
- Nextflow Ambassador
- Cat lover

Introduction

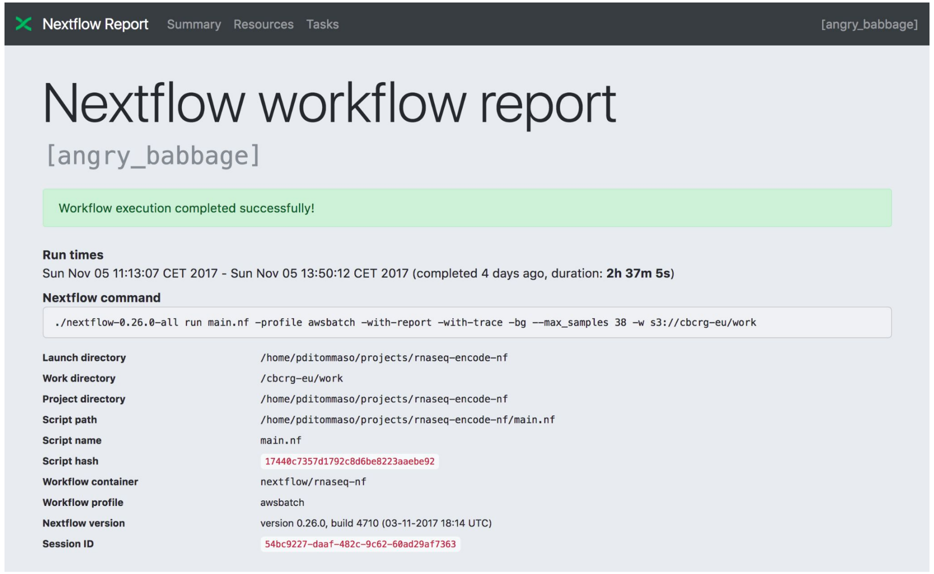
Objectives of today’s session

Reporting

Questions and objectives

- How do I get information about my pipeline run?
- How can I see what commands I ran?
- How can I create a report from my run?

- View Nextflow pipeline run logs.
- Use `nextflow log` to view more information about a specific run.
- Create an HTML report from a pipeline run.



Nextflow workflow report

[angry_babbage]

Workflow execution completed successfully!

Run times

Sun Nov 05 11:13:07 CET 2017 - Sun Nov 05 13:50:12 CET 2017 (completed 4 days ago, duration: **2h 37m 5s**)

Nextflow command

```
./nextflow-0.26.0-all run main.nf -profile awsbatch -with-report -with-trace -bg --max_samples 38 -w s3://cbcrg-eu/work
```

Launch directory	/home/pditommaso/projects/rnaseq-encode-nf
Work directory	/cbcrg-eu/work
Project directory	/home/pditommaso/projects/rnaseq-encode-nf
Script path	/home/pditommaso/projects/rnaseq-encode-nf/main.nf
Script name	main.nf
Script hash	17440c7357d1792c8d6be8223aaebe92
Workflow container	nextflow/rnaseq-nf
Workflow profile	awsbatch
Nextflow version	version 0.26.0, build 4710 (03-11-2017 18:14 UTC)
Session ID	54bc9227-daaf-482c-9c62-60ad29af7363

Config files

Questions and objectives

- What is the difference between the workflow implementation and the workflow configuration?
- How do I configure a Nextflow workflow?
- How do I assign different resources to different processes?
- How do I separate and provide configuration for different computational systems?
- How do I change configuration settings from the default settings provided by the workflow?

- Understand the difference between workflow implementation and configuration.
- Understand the difference between configuring Nextflow and a Nextflow script.
- Create a Nextflow configuration file.
- Understand what a configuration scope is.
- Be able to assign resources to a process.
- Be able to refine configuration settings using process selectors.
- Be able to group configurations into profiles for use with different computer infrastructures.
- Be able to override existing settings.
- Be able to inspect configuration settings before running a workflow.

Pre-requisites

For both reporting and config

You should run the examples on the same directory/environment that you used to carry out the previous days of the training.

Remember! We are learning how to:

- Get reports on previous runs
- Configure runs so they run differently

Documentation can be found here:
https://carpentries-incubator.github.io/workflows-nextflow/instructor/08-reporting.html
https://carpentries-incubator.github.io/workflows-nextflow/instructor/09-configuration.html

Let’s begin!

Reporting

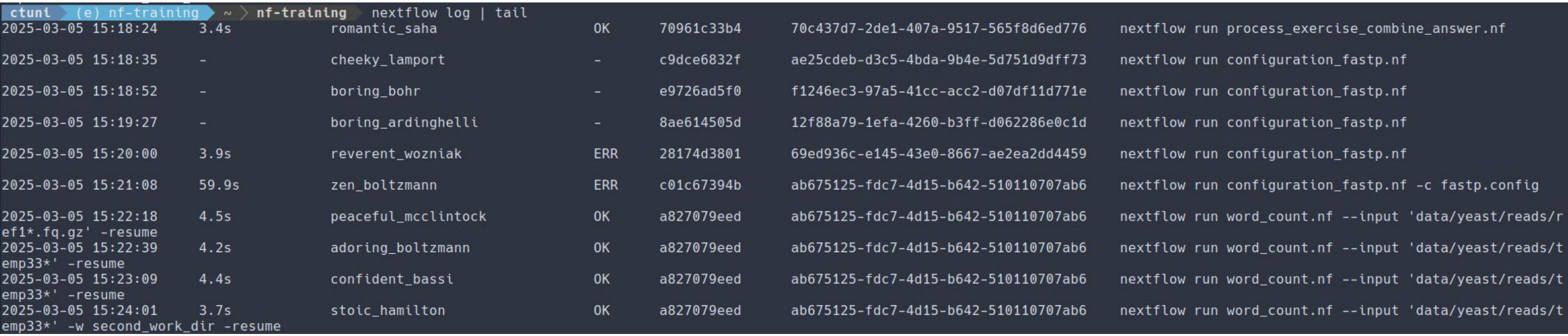
`nextflow log`

Once a script has run, Nextflow stores a log of all the workflows executed in the current folder. Similar to an electronic lab book, this means you have a record of all processing steps and commands run.

You can print Nextflow’s execution history and log information using the `nextflow log` command.

nextflow log

This will print a summary of the executions log and runtime information for all pipelines run. By default, included in the **summary**, are the **date and time** it ran, **how long** it ran for, the **run name**, **run status**, a **revision ID**, the **session id** and the **command run** on the command line.



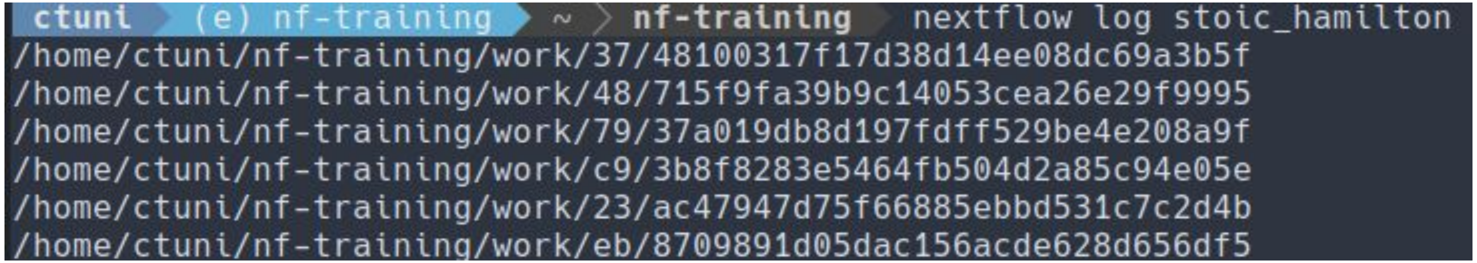
```
 ctuni  (e) nf-training  ~  nf-training   nextflow log | tail
2025-03-05 15:18:24     3.4s            romantic_saha                   OK      70961c33b4      70c437d7-2de1-407a-9517-565f8d6ed776    nextflow run process_exercise_combine_answer.nf

2025-03-05 15:18:35     -               cheeky_lamport                  -       c9dce6832f      ae25cdeb-d3c5-4bda-9b4e-5d751d9dff73    nextflow run configuration_fastp.nf

2025-03-05 15:18:52     -               boring_bohr                     -       e9726ad5f0      f1246ec3-97a5-41cc-acc2-d07df11d771e    nextflow run configuration_fastp.nf

2025-03-05 15:19:27     -               boring_ardinghelli              -       8ae614505d      12f88a79-1efa-4260-b3ff-d062286e0c1d    nextflow run configuration_fastp.nf

2025-03-05 15:20:00     3.9s            reverent_wozniak                ERR     28174d3801      69ed936c-e145-43e0-8667-ae2ea2dd4459    nextflow run configuration_fastp.nf

2025-03-05 15:21:08     59.9s           zen_boltzmann                   ERR     c01c67394b      ab675125-fdc7-4d15-b642-510110707ab6    nextflow run configuration_fastp.nf -c fastp.config

2025-03-05 15:22:18     4.5s            peaceful_mcclintock             OK      a827079eed      ab675125-fdc7-4d15-b642-510110707ab6    nextflow run word_count.nf --input 'data/yeast/reads/r
ef1*.fq.gz' -resume
2025-03-05 15:22:39     4.2s            adoring_boltzmann               OK      a827079eed      ab675125-fdc7-4d15-b642-510110707ab6    nextflow run word_count.nf --input 'data/yeast/reads/t
emp33*' -resume
2025-03-05 15:23:09     4.4s            confident_bassi                 OK      a827079eed      ab675125-fdc7-4d15-b642-510110707ab6    nextflow run word_count.nf --input 'data/yeast/reads/t
emp33*' -resume
2025-03-05 15:24:01     3.7s            stoic_hamilton                  OK      a827079eed      ab675125-fdc7-4d15-b642-510110707ab6    nextflow run word_count.nf --input 'data/yeast/reads/t
emp33*' -w second_work_dir -resume
```

Pipeline execution report

Why do we care about all this information?

If we want to get more information about an individual run we can add the run name or session ID to the `log` command.

This will list the work directory for each process.



```
ctuni  (e) nf-training  ~  nf-training  nextflow log stoic_hamilton
/home/ctuni/nf-training/work/37/48100317f17d38d14ee08dc69a3b5f
/home/ctuni/nf-training/work/48/715f9fa39b9c14053cea26e29f9995
/home/ctuni/nf-training/work/79/37a019db8d197fdff529be4e208a9f
/home/ctuni/nf-training/work/c9/3b8f8283e5464fb504d2a85c94e05e
/home/ctuni/nf-training/work/23/ac47947d75f66885ebbd531c7c2d4b
/home/ctuni/nf-training/work/eb/8709891d05dac156acde628d656df5
```

Those long strings are **taks IDs.** They are a 32 hexadecimal digit,e.g. 3b3485d00b0115f89e4c202eacf82eba. A task's unique ID is generated as a 128-bit hash number obtained from a composition of the task's: **Inputs values, Input files, Command line string, Container ID, Conda environment, Environment modules,Any executed scripts in the bin directory**

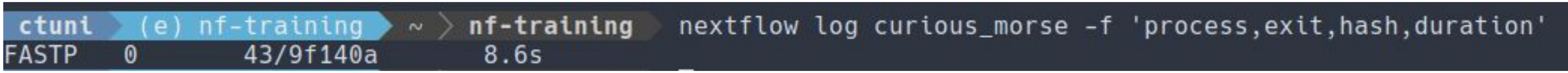
Fields

Controlling what information do we want to see

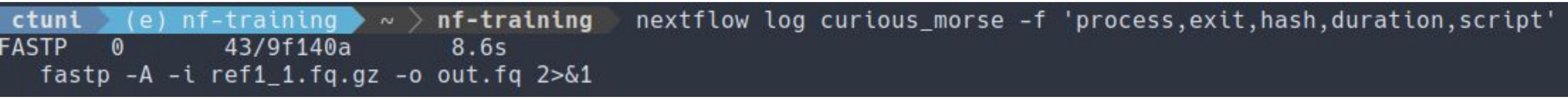
f we want to print more metadata we can use the log command and the option `-f` (fields) followed by a comma-delimited list of fields. This can be composed to track the provenance of a workflow result.

Use `nextflow log -l` to see the complete list of available fields.

If we add the `script` field we'll be able to see the commands executed by the processes.



```
ctuni  (e) nf-training  ~ > nf-training  nextflow log curious_morse -f 'process,exit,hash,duration'
FASTP    0        43/9f140a        8.6s
```



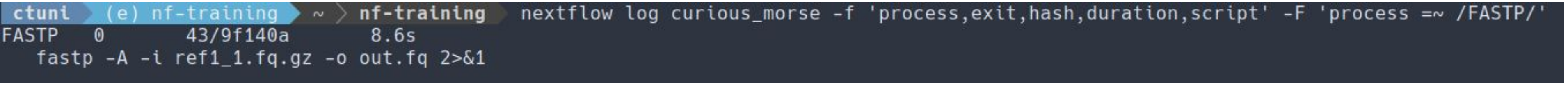
```
ctuni  (e) nf-training  ~ > nf-training  nextflow log curious_morse -f 'process,exit,hash,duration,script'
FASTP    0        43/9f140a        8.6s
    fastp -A -i ref1_1.fq.gz -o out.fq 2>&1
```

Filtering

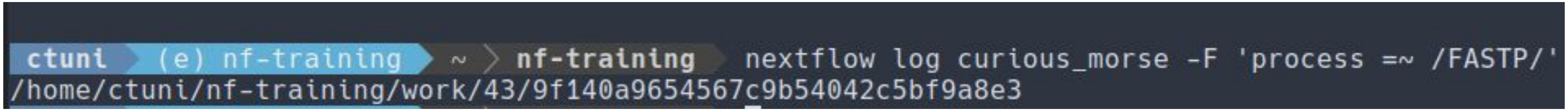
Controlling what information we do NOT want to see

The output from the `log` command can be very long. We can subset the output using the option `-F` (filter) specifying the filtering criteria. This will print only those tasks matching a pattern using the syntax `=~/<pattern>/`.

This can be useful to locate specific tasks work directories.



```
ctuni  (e) nf-training  ~  nf-training  nextflow log curious_morse -f 'process,exit,hash,duration,script' -F 'process =~ /FASTP/'
FASTP   0        43/9f140a       8.6s
   fastp -A -i ref1_1.fq.gz -o out.fq 2>&1
```

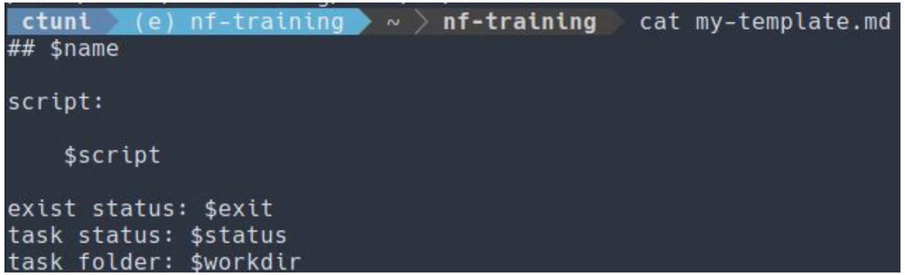


```
ctuni  (e) nf-training  ~  nf-training  nextflow log curious_morse -F 'process =~ /FASTP/'
/home/ctuni/nf-training/work/43/9f140a9654567c9b54042c5bf9a8e3
```

Templates

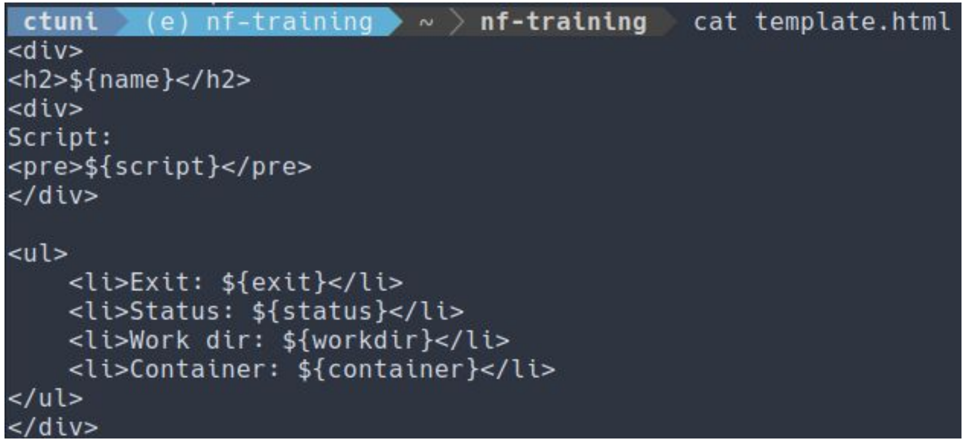
Creating a visually appealing report

The `-t` option allows a template (string or file) to be specified. This makes it possible to create a custom report in any text based format.



```
 ctuni   (e) nf-training  ~ 〉 nf-training   cat my-template.md
## $name

script:

    $script

exist status: $exit
task status: $status
task folder: $workdir
```

This template can be either in markdown (.md) or in HTML (.html)

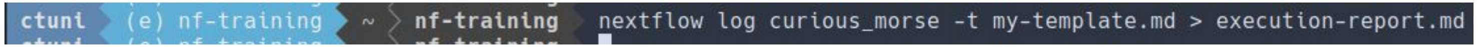


```
 ctuni   (e) nf-training  ~ 〉 nf-training   cat template.html
<div>
<h2>${name}</h2>
<div>
Script:
<pre>${script}</pre>
</div>

<ul>
    <li>Exit: ${exit}</li>
    <li>Status: ${status}</li>
    <li>Work dir: ${workdir}</li>
    <li>Container: ${container}</li>
</ul>
</div>
```

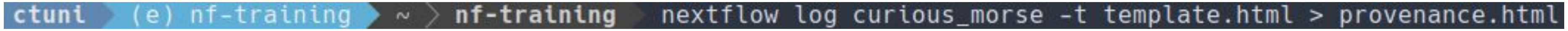
Templates

Creating a visually appealing report

We would use the templates in the following way:



```
ctuni  (e) nf-training  ~  nf-training  nextflow log curious_morse -t my-template.md > execution-report.md
```

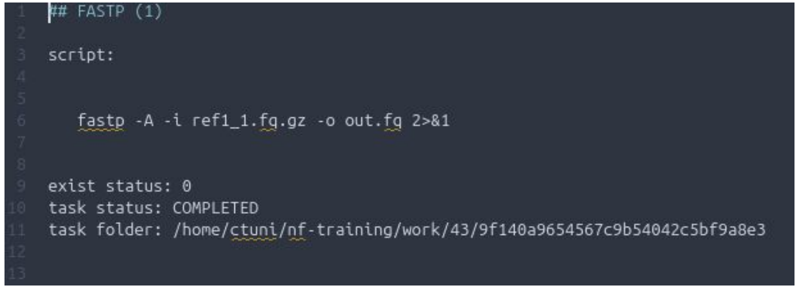


```
ctuni  (e) nf-training  ~  nf-training  nextflow log curious_morse -t template.html > provenance.html
```

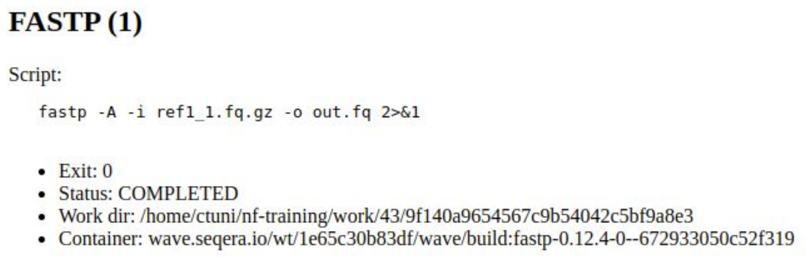
Templates

Creating a visually appealing report

And they would look like this:



```
## FASTP (1)

script:


   fastp -A -i ref1_1.fq.gz -o out.fq 2>&1


exist status: 0
task status: COMPLETED
task folder: /home/ctuni/nf-training/work/43/9f140a9654567c9b54042c5bf9a8e3
```



FASTP (1)

Script:

```
fastp -A -i ref1_1.fq.gz -o out.fq 2>&1
```

- Exit: 0
- Status: COMPLETED
- Work dir: /home/ctuni/nf-training/work/43/9f140a9654567c9b54042c5bf9a8e3
- Container: wave.seqera.io/wt/1e65c30b83df/wave/build:fastp-0.12.4-0--672933050c52f319

Conclusion

Summary, key points, and utility of logs

- Nextflow can produce a custom execution report with run information using the log command.
- You can generate a report using the -t option specifying a template file.
- This is useful for debugging, provenance, and traceability purposes.
- You can know nearly *everything* about your Nexftlow run and its processes using the log capabilities.

Let's continue!

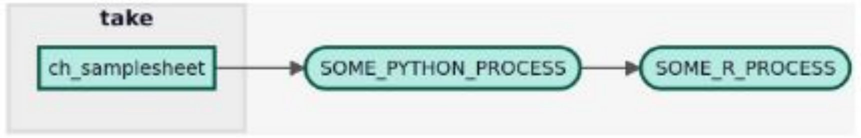
With config files

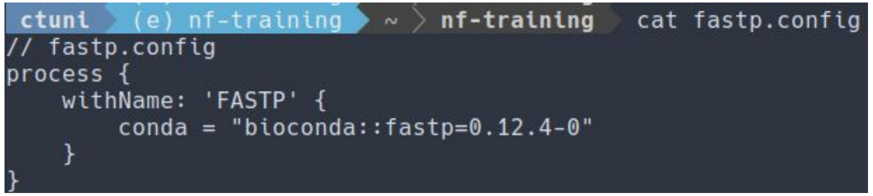
What are config files?

And why are they useful?

A key Nextflow feature is **decoupling:**

- **The workflow**, which describes the flow of data and operations to perform on that data
- **The configuration settings**, required by the underlying execution platform





```
ctuni  (e) nf-training  ~  nf-training  cat fastp.config
// fastp.config
process {
    withName: 'FASTP' {
        conda = "bioconda::fastp=0.12.4-0"
    }
}
```

Antoine Buetti-Dinh

What are config files?

And why are they useful?

This enables the workflow to be **portable**, allowing it to run on different computational platforms such as an institutional **HPC** or cloud infrastructure, **without needing to modify the workflow implementation.**

We have seen earlier that it is possible to provide a process with **directives**. These directives are **process specific configuration settings**. Similarly, we have also provided **parameters** to our workflow which are **parameter configuration settings**. These configuration settings **can be separated from the workflow implementation, into a configuration file.**

Getting to know configs

Settings in a configuration file are sets of name-value pairs `(name = value)`. The name is a specific property to set, while the value can be anything you can assign to a variable, for example, strings, booleans, or other variables. It is also possible to access any variable defined in the host environment such as `$PATH`, `$HOME`, `$PWD`, etc.

```
// nextflow.config
my_home_dir = "$HOME"
```

Accessing variables in your configuration file

Important to know

Generally, variables and functions defined in a configuration file **are not accessible from the workflow script**. Only variables defined using the **params scope and the env scope** (without env prefix) can be accessed from the workflow script.

```
workflow {
    MY_PROCESS( params.input )
}
```

Scopes

Governing behaviour of different elements

Settings are also partitioned into scopes, which govern the behaviour of different elements of the workflow. For example, **workflow parameters are governed from the params scope**, while **process directives are governed from the process scope**. A full list of the available scopes can be found in the documentation. **It is also possible to define your own scope.**

Configuration settings for a **workflow are often stored in the file nextflow.config** which is in the same directory as the workflow script.

Let’s write some config!

Dot and brace notation

Configuration can be written in either of two ways. The first is using dot notation, and the second is using brace notation. Both forms of notation can be used in the same configuration file.

Dot notation:

```
params.input = ''                  // The
workflow parameter "input" is assigned an
empty string to use as a default value
params.outdir = './results'    // The
workflow parameter "outdir" is assigned the
value './results' to use by default.
```

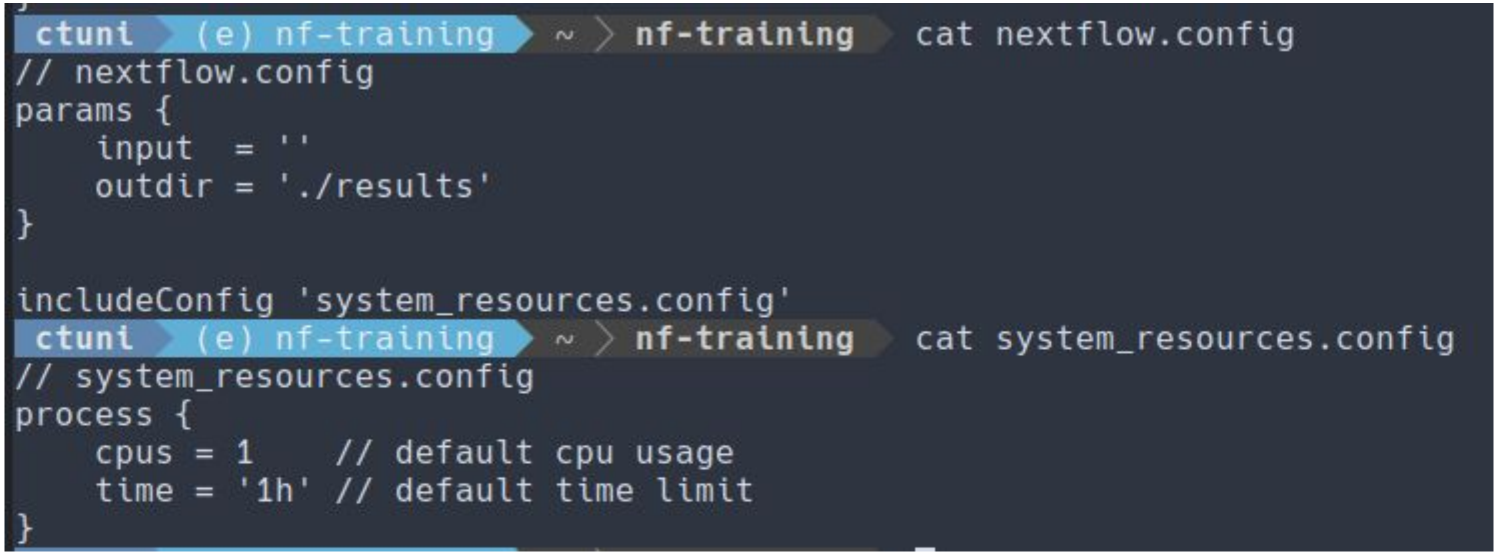
Brace notation:

```
params {
    input  = ''
    outdir = './results'
}
```

includeConfig

Keeping configs separate

Configuration files can also be separated into multiple files and i**ncluded into another using the includeConfig statement.**



```
 ctuni  (e) nf-training  ~ 〉nf-training   cat nextflow.config
// nextflow.config
params {
    input  = ''
    outdir = './results'
}

includeConfig 'system_resources.config'
 ctuni  (e) nf-training  ~ 〉nf-training   cat system_resources.config
// system_resources.config
process {
    cpus = 1    // default cpu usage
    time = '1h' // default time limit
}
```

Config prioritisation

Managing several config files without going crazy

Configuration settings can be spread across several files. This also allows settings to be overridden by other configuration files. The priority of a setting is determined by the following order, ranked from highest to lowest.

1. Parameters specified on the command line (`--param_name value`).
2. Parameters provided using the `-params-file` option.
3. Config file specified using the `-c` my_config option.
4. The config file named `nextflow.config` in the current directory.
5. The config file named `nextflow.config` in the workflow project directory (`$projectDir`: the directory where the script to be run is located).
6. The config file `$HOME/.nextflow/config`
7. Values defined within the workflow script itself (e.g., `main.nf`).

If configuration is provided by more than one of these methods, configuration is merged giving higher priority to configuration provided higher in the list.

Existing configuration can be completely ignored by using `-C <custom.config>` to use only configuration provided in the `custom.config` file.

Configuring Nextflow vs Configuring a Nextflow workflow

One dash or two?

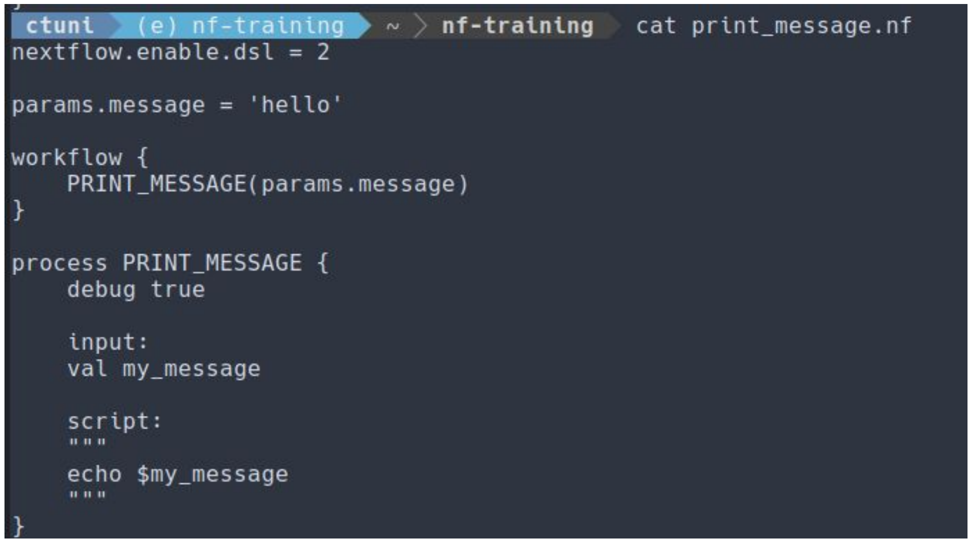
Parameters starting with a single dash – (e.g., `-c my_config.config`) are **configuration options for nextflow**, while parameters starting with a double dash `--` (e.g., `--outdir`) are **workflow parameters** defined in the params scope.

The majority of Nextflow configuration settings must be provided on the command-line, however a handful of settings can also be provided within a configuration file, such as `workdir = '/path/to/work/dir'` (`-w /path/to/work/dir`) or `resume = true` (`-resume`), and do not belong to a configuration scope.

Determining script output

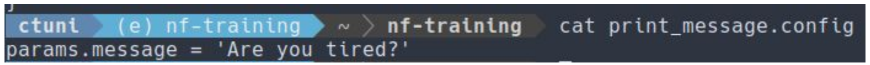
What will happen?

Given the following script:



```
ctuni  (e) nf-training  ~  nf-training  cat print_message.nf
nextflow.enable.dsl = 2

params.message = 'hello'

workflow {
    PRINT_MESSAGE(params.message)
}

process PRINT_MESSAGE {
    debug true

    input:
    val my_message

    script:
    """
    echo $my_message
    """
}
```

And the following config:

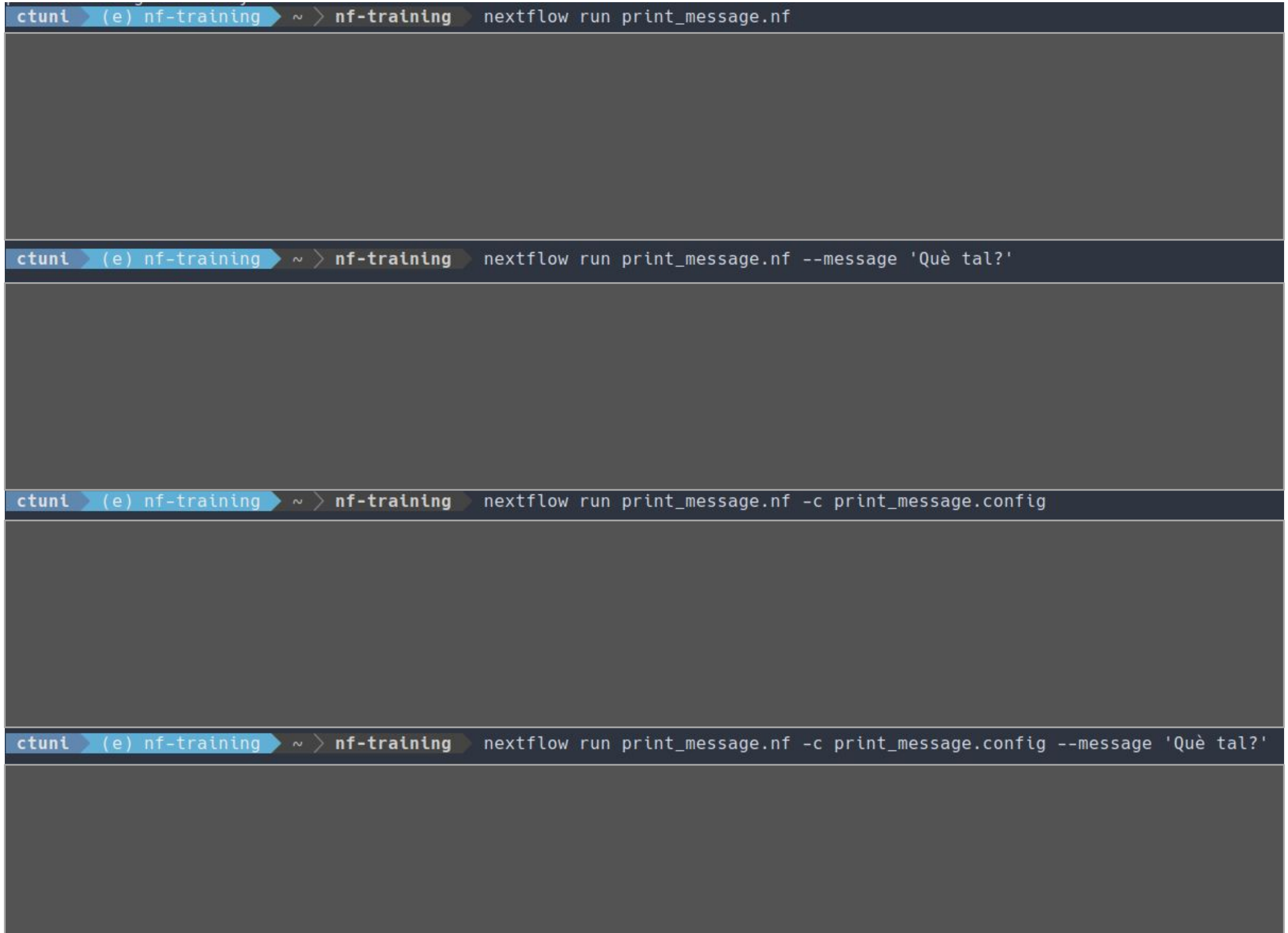


```
ctuni  (e) nf-training  ~  nf-training  cat print_message.config
params.message = 'Are you tired?'
```

Determining script output

What will happen?

What will happen if I run the following commands?



```
ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf
```

```
ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf --message 'Què tal?'
```

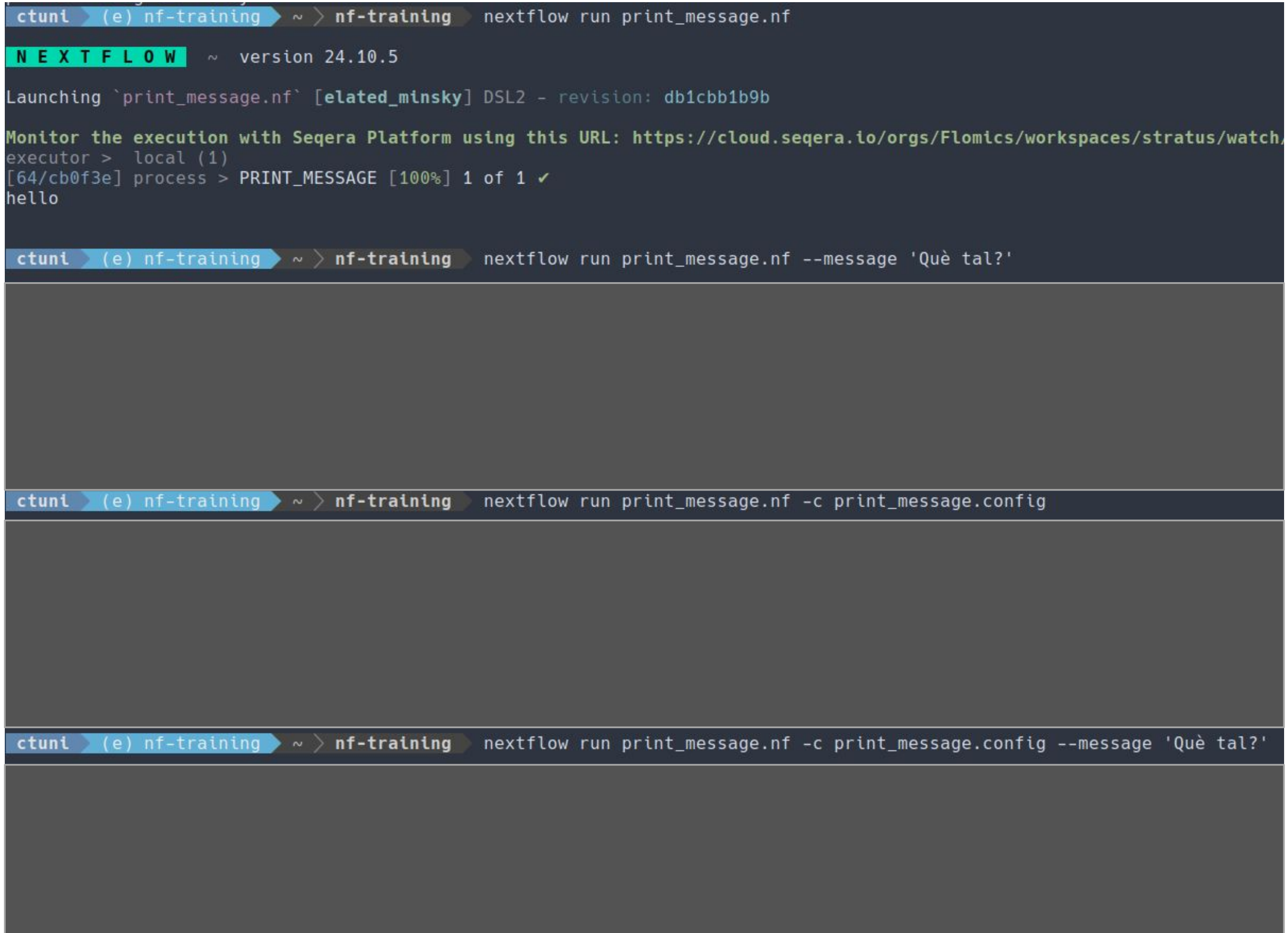
```
ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf -c print_message.config
```

```
ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf -c print_message.config --message 'Què tal?'
```

Determining script output

What will happen?

Workflow script **uses the value in print_message.nf**

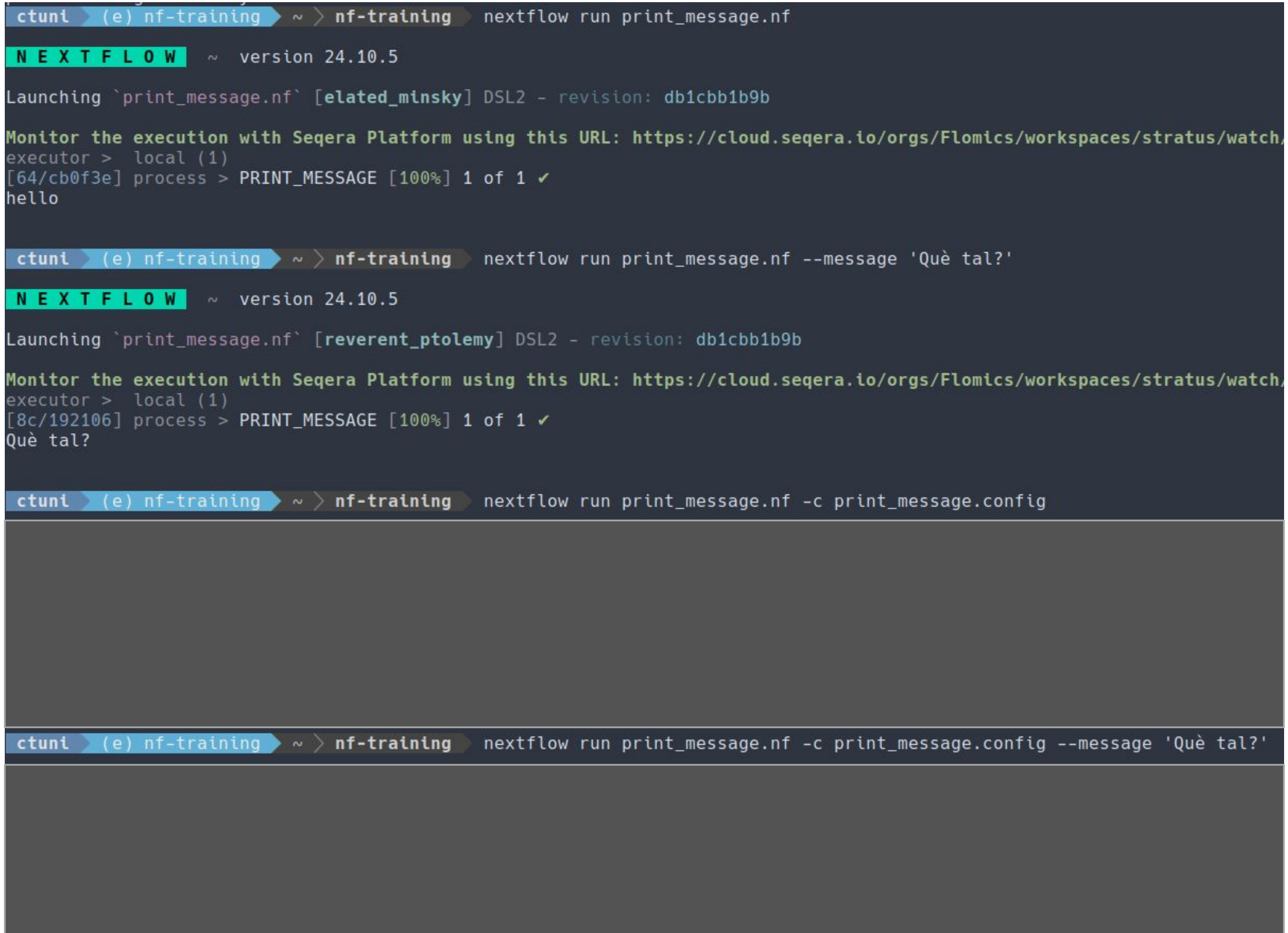


```
ctuni  (e) nf-training  ~ > nf-training  nextflow run print_message.nf

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [elated_minsky] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch
executor >  local (1)
[64/cb0f3e] process > PRINT_MESSAGE [100%] 1 of 1 ✔
hello

ctuni  (e) nf-training  ~ > nf-training  nextflow run print_message.nf --message 'Què tal?'

ctuni  (e) nf-training  ~ > nf-training  nextflow run print_message.nf -c print_message.config

ctuni  (e) nf-training  ~ > nf-training  nextflow run print_message.nf -c print_message.config --message 'Què tal?'
```

Determining script output

What will happen?

The command-line parameter **overrides the script setting.**

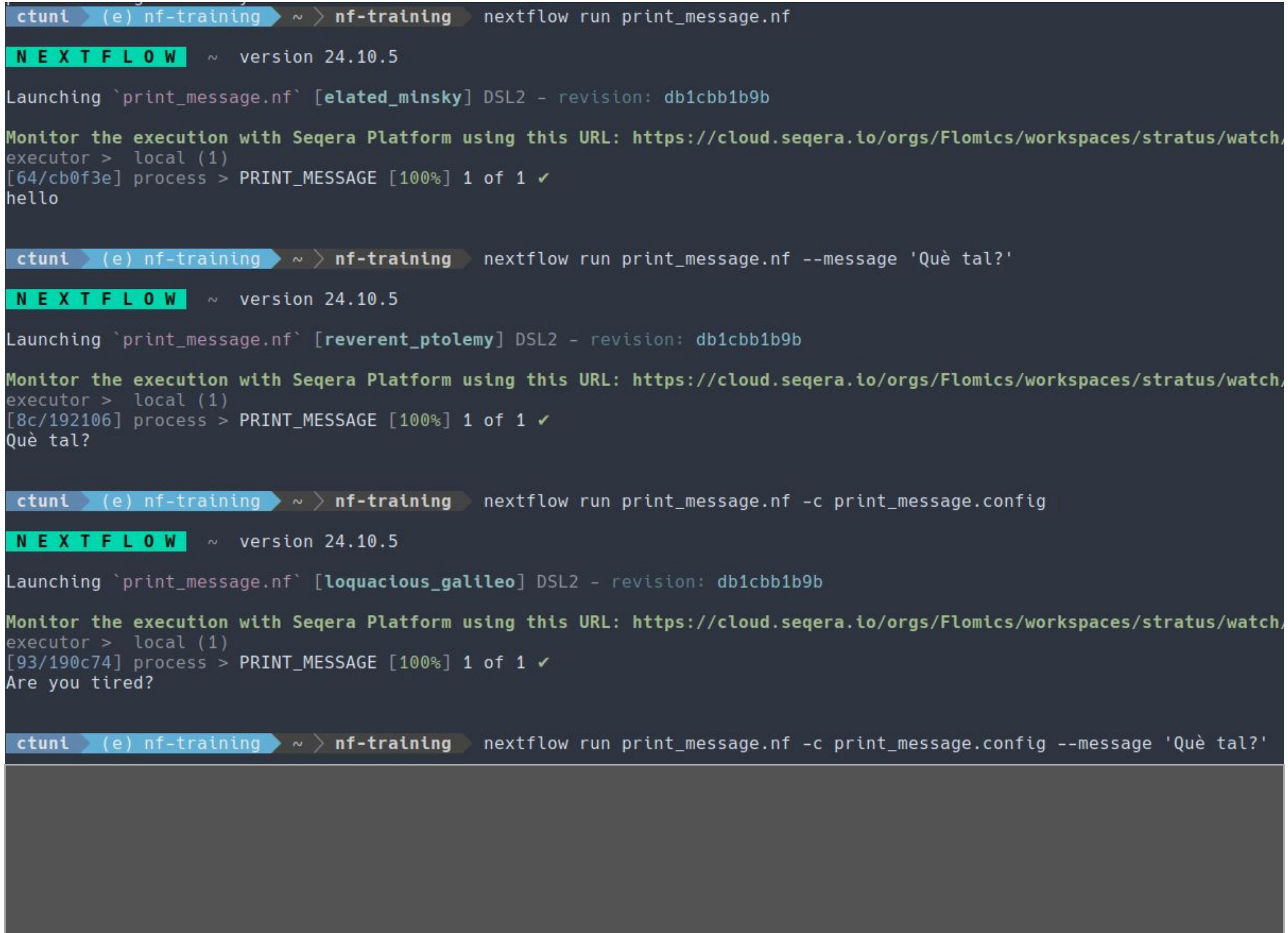


```
ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [elated_minsky] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[64/cb0f3e] process > PRINT_MESSAGE [100%] 1 of 1 ✔
hello

ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf --message 'Què tal?'

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [reverent_ptolemy] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[8c/192106] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Què tal?

ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf -c print_message.config

ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf -c print_message.config --message 'Què tal?'
```

Determining script output

What will happen?

The configuration **overrides the script setting**

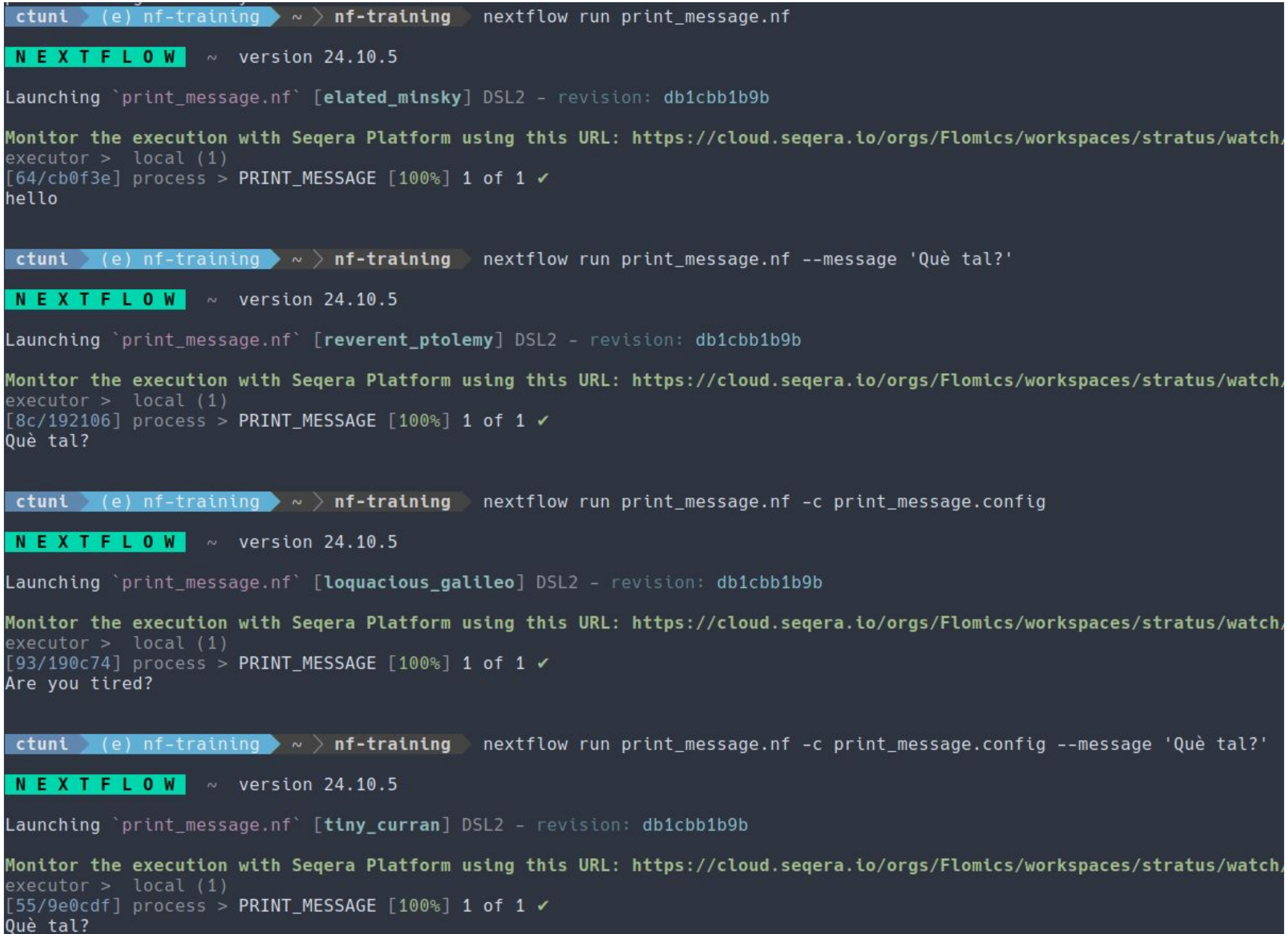


```
ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [elated_minsky] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[64/cb0f3e] process > PRINT_MESSAGE [100%] 1 of 1 ✔
hello


ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf --message 'Què tal?'

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [reverent_ptolemy] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[8c/192106] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Què tal?


ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf -c print_message.config

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [loquacious_galileo] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[93/190c74] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Are you tired?


ctuni  (e) nf-training  ~ > nf-training   nextflow run print_message.nf -c print_message.config --message 'Què tal?'
```

Determining script output

What will happen?

The command-line parameter **overrides both the script and configuration settings.**



```
ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [elated_minsky] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[64/cb0f3e] process > PRINT_MESSAGE [100%] 1 of 1 ✔
hello

ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf --message 'Què tal?'

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [reverent_ptolemy] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[8c/192106] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Què tal?

ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf -c print_message.config

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [loquacious_galileo] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[93/190c74] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Are you tired?

ctuni  (e) nf-training  ~  nf-training   nextflow run print_message.nf -c print_message.config --message 'Què tal?'

 N E X T F L O W   ~  version 24.10.5

Launching `print_message.nf` [tiny_curran] DSL2 - revision: db1cbb1b9b

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (1)
[55/9e0cdf] process > PRINT_MESSAGE [100%] 1 of 1 ✔
Què tal?
```

Configuring process behaviour

The `process` scope

Earlier we saw that `process` directives allow the specification of settings for the task execution such as `cpus`, `memory`, `conda` and other resources in the pipeline script. This is useful when prototyping a small workflow script, however this ties the configuration to the workflow, **making it less portable**. **A good practice is to separate the process configuration settings into another file.**

The **process** configuration scope allows the setting of any process directives in the Nextflow configuration file.

```
// nextflow.config
process {
     cpus = 2
     memory = 8.GB
     time = '1 hour'
     publishDir = [ path:
params.outdir, mode: 'copy' ]
}
```

Unit values

How to use units that Nextflow understands

Memory and time duration units can be specified either using a **string based notation** in which the digit(s) and the unit can be separated by a space character, or by using the **numeric notation** in which the digit(s) and the unit are separated by a dot character and not enclosed by quote characters.

These settings are applied to all processes in the workflow. **A process selector can be used to apply the configuration to a specific process or group of processes.**

String	Numeric	Value
'10.KB'	10.KB	10240 bytes
'500.MB'	500.MB	524288000 bytes
' 1 min'	1.min	60 seconds
'1 hour 25 sec'		1 hour and 25 seconds

Process selectors

Controlling a named process

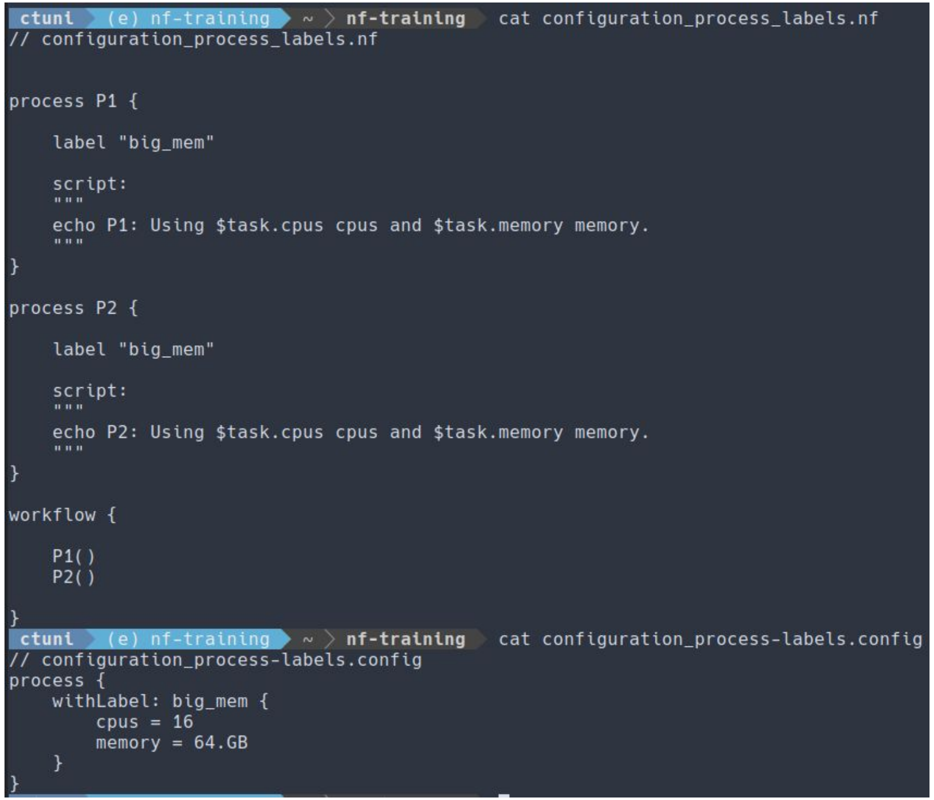
The resources for a specific process can be defined using `withName`: followed by the process name e.g., '`FASTQC`', and the directives within curly braces. For example, we can specify different `cpus` and `memory` resources for the processes `INDEX` and `FASTQC` as follows:

```
// process_resources.config
process {
      withName: INDEX {
      cpus = 4
      memory = 8.GB
      }
      withName: FASTQC {
      cpus = 2
      memory = 4.GB
      }
}
```

Process selectors

Controlling a named process

When a workflow has many processes, **it is inconvenient to specify directives for all processes individually**, especially if directives are repeated for groups of processes. A helpful strategy is to annotate the processes using the **label** directive (processes can have multiple labels). The **withLabel** selector then allows the configuration of all processes annotated with a specific label, as show:



```
ctuni  (e) nf-training  ~ > nf-training   cat configuration_process_labels.nf
// configuration_process_labels.nf


process P1 {

    label "big_mem"

    script:
    """
    echo P1: Using $task.cpus cpus and $task.memory memory.
    """
}

process P2 {

    label "big_mem"

    script:
    """
    echo P2: Using $task.cpus cpus and $task.memory memory.
    """
}

workflow {

    P1()
    P2()

}
ctuni  (e) nf-training  ~ > nf-training   cat configuration_process-labels.config
// configuration_process-labels.config
process {
    withLabel: big_mem {
        cpus = 16
        memory = 64.GB
    }
}
```

Process selectors

Controlling everything!

Another strategy is to use process selector expressions. Both withName: and withLabel: allow the use of regular expressions to apply the same configuration to all processes matching a pattern. Regular expressions must be quoted, unlike simple process names or labels.

A regular expression cheat-sheet can be found here if you would like to write more expressive expressions: https://www.jrebel.com/system/files/regular-expressions-cheat-sheet.pdf

- The `|` matches either-or, e.g., `withName: 'INDEX|FASTQC'` applies the configuration to any process matching the name `INDEX` **or** `FASTQC`.
- The `!` **inverts** a selector, e.g., `withLabel: '!small_mem'` applies the configuration to any process without the `small_mem` label.
- The `.*` matches **any number of characters**, e.g., `withName: 'NFCORE_RNASEQ:RNA_SEQ:BAM_SORT:.*'` matches all processes of the workflow `NFCORE_RNASEQ:RNA_SEQ:BAM_SORT`

Selector priority

Managing several selectors without going crazy

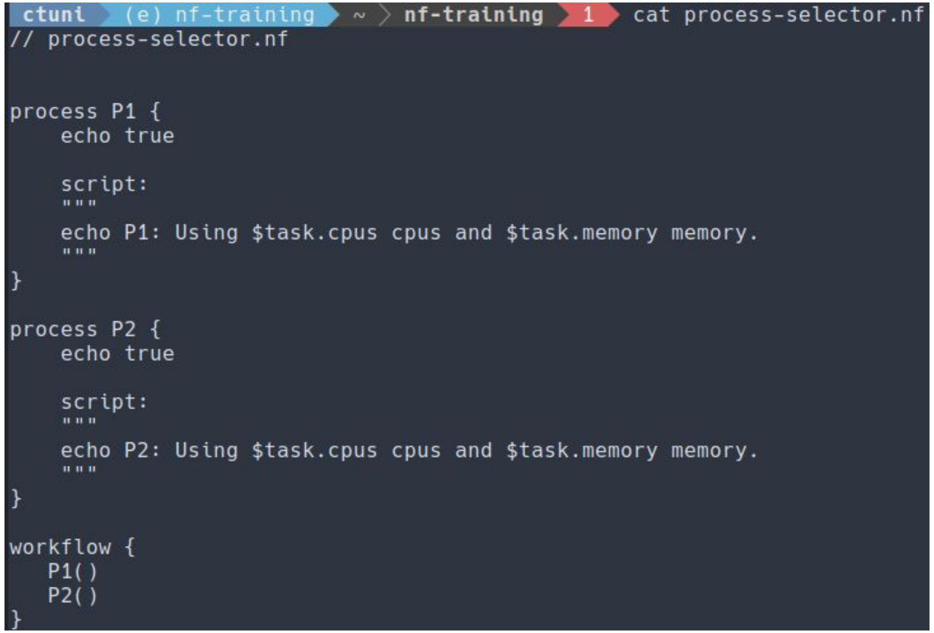
When mixing generic process configuration and selectors, the following priority rules are applied (from highest to lowest):

- withName selector definition.
- withLabel selector definition.
- Process specific directive defined in the workflow script.
- Process generic process configuration.

Hands-on example

The best way of learning is doing

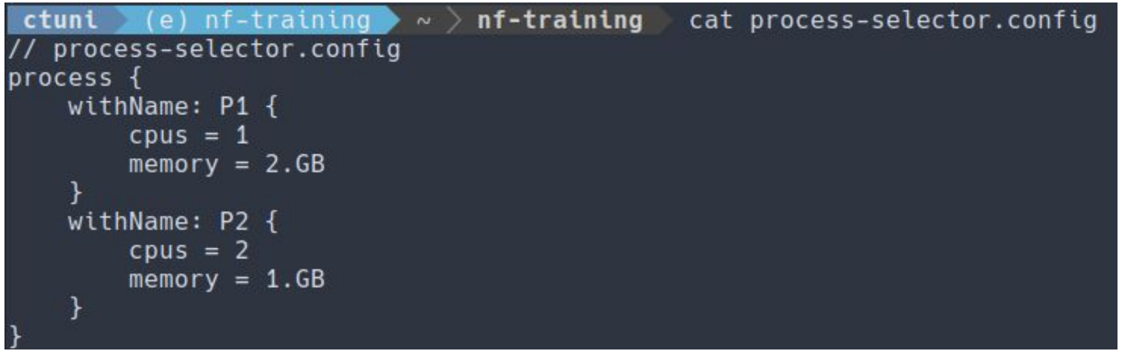
Create a Nextflow config, `process-selector.config`, specifying different `cpus` and `memory` resources for the two processes `P1` (cpus 1 and memory 2.GB) and `P2` (cpus 2 and memory 1.GB), where `P1` and `P2` are defined as follows:



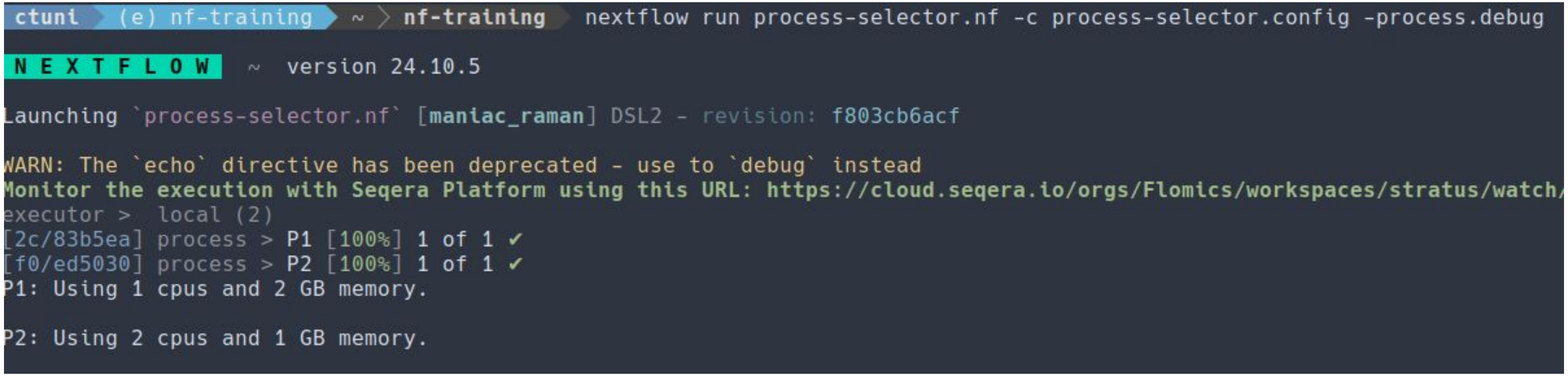
```
ctuni  (e) nf-training  ~  nf-training  1  cat process-selector.nf
// process-selector.nf


process P1 {
    echo true

    script:
    """
    echo P1: Using $task.cpus cpus and $task.memory memory.
    """
}

process P2 {
    echo true

    script:
    """
    echo P2: Using $task.cpus cpus and $task.memory memory.
    """
}

workflow {
   P1()
   P2()
}
```

Hands-on example

Solution



```
 ctuni  (e) nf-training  ~  nf-training   cat process-selector.config
// process-selector.config
process {
    withName: P1 {
        cpus = 1
        memory = 2.GB
    }
    withName: P2 {
        cpus = 2
        memory = 1.GB
    }
}
```



```
 ctuni  (e) nf-training  ~  nf-training   nextflow run process-selector.nf -c process-selector.config -process.debug

 N E X T F L O W   ~  version 24.10.5

Launching `process-selector.nf` [maniac_raman] DSL2 - revision: f803cb6acf

WARN: The `echo` directive has been deprecated - use to `debug` instead
Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/workspaces/stratus/watch/
executor >  local (2)
[2c/83b5ea] process > P1 [100%] 1 of 1 ✔
[f0/ed5030] process > P2 [100%] 1 of 1 ✔
P1: Using 1 cpus and 2 GB memory.

P2: Using 2 cpus and 1 GB memory.
```

Dynamic expressions

Even more control!

A common scenario is that configuration settings may **depend on the data being processed**. Such settings can be **dynamically expressed using a closure**. For example, we can specify the **memory** required as a multiple of the number of **cpus**. Similarly, we can publish results to a subfolder based on the sample name.

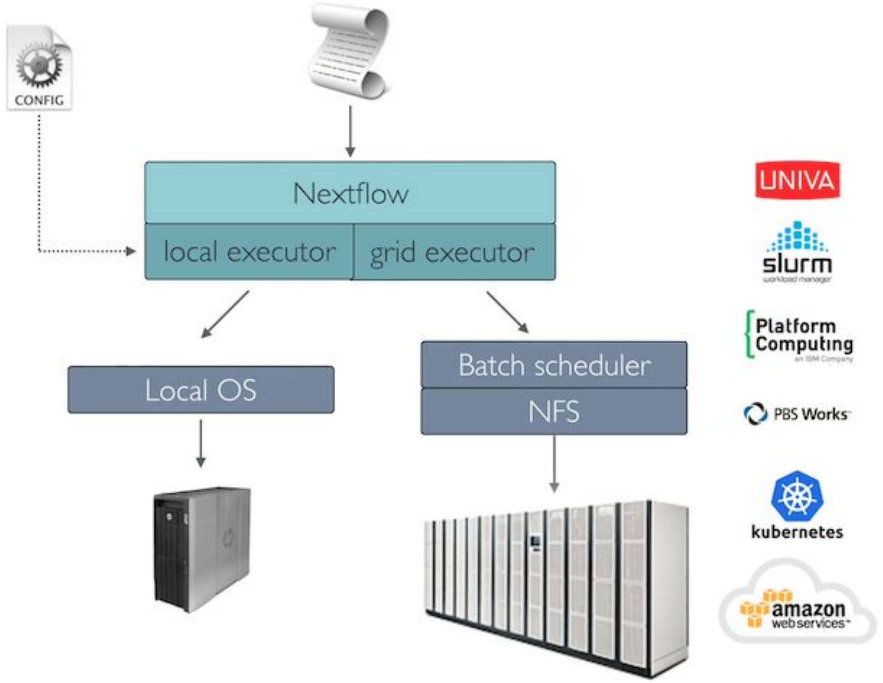
```
process FASTQC {

      input:
      tuple val(sample), path(reads)

      script:
      """
      fastqc -t $task.cpus $reads
      """
}

// nextflow.config
process {
      withName: FASTQC {
      cpus = 2
      memory = { 2.GB * task.cpus }
      publishDir = { "fastqc/$sample" }
      }
}
```

Configuring execution platforms

Local or in the cloud?

Nextflow supports a wide range of execution platforms, from running locally, to running on HPC clusters or cloud infrastructures. See https://www.nextflow.io/docs/latest/executor.html for the full list of supported executors.

Image description: A diagram illustrating the different executors available in Nextflow. The diagram shows a configuration file feeding into Nextflow, which has both local and grid executors. The local executor connects to the Local OS and a standalone computer, while the grid executor connects to a batch scheduler and NFS, which further connects to various computing resources such as UNIVA, SLURM, Platform Computing, PBS Works, Kubernetes, and Amazon Web Services.



Configuring execution platforms

Local or in the cloud?

The default executor configuration is defined within the executor scope (https://www.nextflow.io/docs/latest/config.html#scope-executor). For example, in the config below we specify the executor as Sun Grid Engine, `sge` and the number of tasks the executor will handle in a parallel manner (`queueSize`) to 10.
The `process.executor` directive allows you to override the executor to be used by a specific process. This can be useful, for example, when there are short running tasks that can be run locally, and are unsuitable for submission to HPC executors (check for guidelines on best practice use of your execution system). Other process directives such as `process.clusterOptions`, `process.queue`, and `process.machineType` can be also be used to further configure processes depending on the executor used.

```
//nextflow.config
executor {
    name = 'sge'
    queueSize = 10
}
process {
    withLabel: 'short' {
    executor = 'local'
    }
}
```

Configuring software requirements

Support for different technologies

An important feature of Nextflow is the ability to manage software using different technologies. It supports the **Conda** package management system, and container engines such as **Docker**, **Singularity**, **Podman**, **Charliecloud**, and **Shifter**. These technologies allow one to package tools and their dependencies into a software environment such that the **tools will always work as long as the environment can be loaded.**

This facilitates **portable and reproducible workflows**. Software environment specification is managed from the `process` scope, allowing the use of process selectors to manage which processes load which software environment. Each technology also has its own scope to provide further technology specific configuration settings.

A hands-on example

Software configuration using Conda

Conda is a software package and environment management system that runs on Linux, Windows, and Mac OS. Software packages are **bundled into Conda environments along with their dependencies for a particular operating system** (Not all software is supported on all operating systems). Software packages are tied to conda channels, for example, bioinformatic software packages are found and installed from the BioConda channel.

A Conda environment can be configured in several ways:

- Provide a path to an existing Conda environment.
- Provide a path to a Conda environment specification file (written in YAML).
- Specify the software package(s) using the `<channel>::<package_name>=<version>` syntax (separated by spaces), which then builds the Conda environment when the process is run.

A hands-on example

Software configuration using Conda

```
process {
    conda =
"/home/user/miniconda3/envs/my_conda
_env"
    withName: FASTQC {
    conda = "environment.yml"
    }
    withName: SALMON {
    conda = "bioconda::salmon=1.5.2"
    }
}
```

There is an optional `conda` scope which allows you to control the creation of a Conda environment by the Conda package manager. For example, `conda.cacheDir` specifies the path where the Conda environments are stored. By default this is in `conda` folder of the work directory.

A hands-on example

Software configuration using Conda

We are going to define a software requirement using the “`conda`” directive.

The software we are going to use is `fastp`, a tool used for fast processing of next-generation sequencing data (like RNA or DNA sequences).

Each time the process is called we are going to run `fastp -A -i ${read} -o out.fq 2>&1`

1. Create a config file for the Nextflow script `configuration_fastp.nf` .
2. Add a `conda` directive for the process name `FASTP` that includes the `bioconda` package `fastp`, version `0.12.4-0`.
3. **Hint** You can specify the `conda` packages using the syntax `<channel>::<package_name>=<version>` e.g. `bioconda::salmon=1.5.2`
4. Run the Nextflow script `configuration_fastp.nf` with the configuration file using the `-c` option.

A hands-on example

Software configuration using Conda

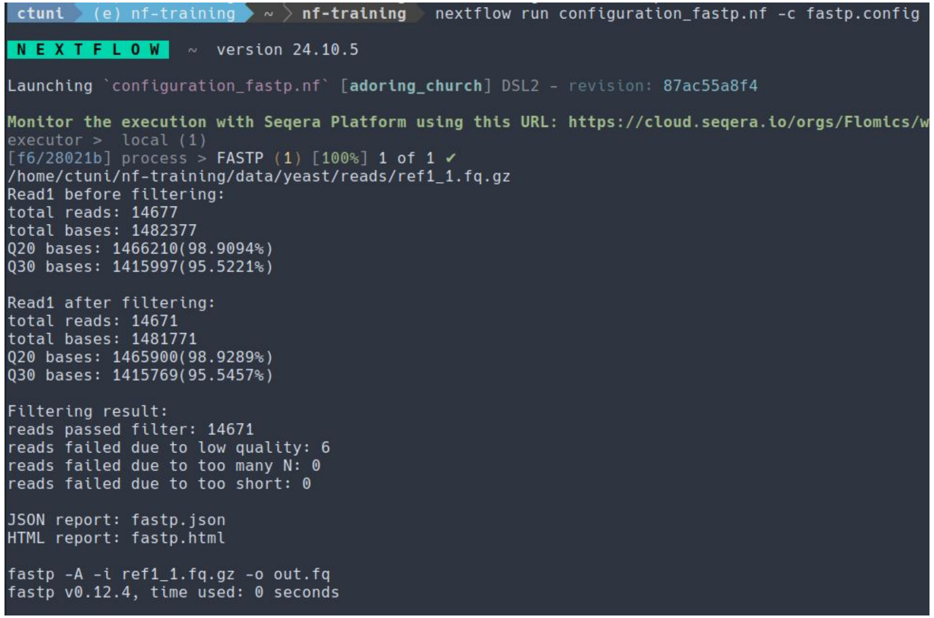
```
// configuration_fastp.nf
process FASTP {

   input:
   path read

   output:
   stdout

   script:
   """
   fastp -A -i ${read} -o out.fq 2>&1
   """
}

workflow {
      FASTP(
Channel.fromPath('data/yeast/reads/ref1_1.fq.gz').vie
w() ).view()
}
```

1. Create a config file for the Nextflow script `configuration_fastp.nf`.
2. Add a `conda` directive for the process name `FASTP` that includes the `bioconda` package `fastp`, version `0.12.4-0`.
3. **Hint** You can specify the `conda` packages using the syntax `<channel>::<package_name>=<version>` e.g. `bioconda::salmon=1.5.2`
4. Run the Nextflow script `configuration_fastp.nf` with the configuration file using the `-c` option.

A hands-on example

Software configuration using Conda

```
// fastp.config
process {
     withName: 'FASTP' {
     conda =
"bioconda::fastp=0.12.4-0"
     }
}
```



```
ctuni  (e) nf-training  ~ > nf-training  nextflow run configuration_fastp.nf -c fastp.config

N E X T F L O W  ~  version 24.10.5

Launching `configuration_fastp.nf` [adoring_church] DSL2 - revision: 87ac55a8f4

Monitor the execution with Seqera Platform using this URL: https://cloud.seqera.io/orgs/Flomics/w
executor >  local (1)
[f6/28021b] process > FASTP (1) [100%] 1 of 1 ✔
/home/ctuni/nf-training/data/yeast/reads/ref1_1.fq.gz
Read1 before filtering:
total reads: 14677
total bases: 1482377
Q20 bases: 1466210(98.9094%)
Q30 bases: 1415997(95.5221%)

Read1 after filtering:
total reads: 14671
total bases: 1481771
Q20 bases: 1465900(98.9289%)
Q30 bases: 1415769(95.5457%)

Filtering result:
reads passed filter: 14671
reads failed due to low quality: 6
reads failed due to too many N: 0
reads failed due to too short: 0

JSON report: fastp.json
HTML report: fastp.html

fastp -A -i ref1_1.fq.gz -o out.fq
fastp v0.12.4, time used: 0 seconds
```

A hands-on example

Software configuration using Docker

Docker is a container technology. Container images are lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Containerized software is intended to run the same regardless of the underlying infrastructure, unlike other package management technologies which are operating system dependant (See the published article on Nextflow). For each container image used, Nextflow uses Docker to spawn an and isolated container instance for process task.

To use Docker, we must provide a container image path using the `process.container` directive, and also enable docker in the docker scope, `docker.enabled = true`. A container image path takes the form `(protocol://)registry/repository/image:version--build`. By default, Docker containers run software using a privileged user. This can cause issues, and so it is also a good idea to supply your user and group via the `docker.runOptions`.

A hands-on example

Software configuration using Docker

The Docker run option `-u $(id -u):$(id -g)` is used to specify the user and group IDs (UID and GID) that the container should use when running. This option ensures that the processes inside the container run with the same UID and GID as the user executing the Docker command on the host system.

```
process.container =
'quay.io/biocontainers/salmon:1.5.2-
-h84f40af_0'
docker.enabled = true
docker.runOptions = '-u $(id
-u):$(id -g)'
```

A hands-on example

Software configuration using Singularity

Singularity is another container technology, commonly used on HPC clusters. It is different to Docker in several ways. The primary differences are that processes are run as the user, and certain directories are automatically “mounted” (made available) in the container instance. Singularity also supports building Singularity images from Docker images, allowing Docker image paths to be used as values for `process.container`.

Singularity is enabled similarly to Docker. A container image path must be provided using process.container and singularity enabled using `singularity.enabled = true`.

```
process.container =
'https://depot.galaxyproject.org/singularity/salmon:1.5.2--h84f40af_0'
singularity.enabled = true
```

Container protocols

All supported by Nextflow

- `docker://`: download the container image from the Docker Hub and convert it to the Singularity format (default).
- `library://`: download the container image from the Singularity Library service.
- `shub://`: download the container image from the Singularity Hub.
- `docker-daemon://`: pull the container image from a local Docker installation and convert it to a Singularity image file.
- `https://`: download the singularity image from the given URL.
- `file://`: use a singularity image on local computer storage.

Configuration profiles

Keeping everything tidy

One of the most powerful features of Nextflow configuration is to predefine multiple configurations or profiles for different execution platforms. This allows a group of predefined settings to be called with a short invocation, `-profile <profile name>`.

Configuration profiles are defined in the `profiles` scope, which group the attributes that belong to the same profile using a common prefix.

```
//configuration_profiles.config
profiles {

       standard {
       params.genome = '/local/path/ref.fasta'
       process.executor = 'local'
       }

       cluster {
       params.genome = '/data/stared/ref.fasta'
       process.executor = 'sge'
       process.queue = 'long'
       process.memory = '10GB'
       process.conda = '/some/path/env.yml'
       }

       cloud {
       params.genome = '/data/stared/ref.fasta'
       process.executor = 'awsbatch'
       process.container = 'cbcrg/imagex'
       docker.enabled = true
       }

}
```

Configuration profiles

Keeping everything tidy

This configuration defines three different profiles: `standard`, `cluster` and `cloud` that set different process configuration strategies depending on the target execution platform. By convention, the standard profile is implicitly used when the user specifies no other profile. To enable a specific profile use `-profile` option followed by the profile name:

```
nextflow run <your script> -profile
cluster
```

```
//configuration_profiles.config
profiles {

        standard {
        params.genome = '/local/path/ref.fasta'
        process.executor = 'local'
        }

        cluster {
        params.genome = '/data/stared/ref.fasta'
        process.executor = 'sge'
        process.queue = 'long'
        process.memory = '10GB'
        process.conda = '/some/path/env.yml'
        }

        cloud {
        params.genome = '/data/stared/ref.fasta'
        process.executor = 'awsbatch'
        process.container = 'cbcrg/imagex'
        docker.enabled = true
        }

}
```

Configuration order

Something to keep in mind

Settings from profiles will override general settings in the configuration file. However, it is also important to remember that configuration is evaluated in the order it is read in. For example, in the following example, the `publishDir` directive will always take the value `'results'` even when the profile `hpc` is used. This is because the setting is evaluated before Nextflow knows about the hpc profile. If the `publishDir` directive is moved to after the profiles scope, then `publishDir` will use the correct value of `params.results`.

```
params.results = 'results'
process.publishDir = params.results
profiles {
    hpc {
    params.results =
'/long/term/storage/results'
    }
}
```

Conclusion

Summary, key points, and utility of configs

- Nextflow configuration can be managed using a Nextflow configuration file.
- Nextflow configuration files are plain text files containing a set of properties.
- You can define process specific settings, such as cpus and memory, within the process scope.
- You can assign different resources to different processes using the process selectors `withName` or `withLabel`.
- You can define a profile for different configurations using the `profiles` scope. These profiles can be selected when launching a pipeline execution by using the `-profile` command-line option
- Nextflow configuration settings are evaluated in the order they are read-in.

nextflow

See you on Monday 17th!

Thank you