Introduction to Nextflow

Yuwu Chen

LSU

Outline

What is Nextflow

- Nextflow is a workflow framework for
  - scalable and reproducible scientific computations;
  - integrating bioinformatics scripts into pipelines;
  - extending Unix pipes model

- Nextflow is also a platform for the development and implementation of new pipelines.
- The first Github version 0.2.2 was released in 2013
- Latest stable version is 21.04.0 released on May 2, 2021

Features of Nextflow

- Fast prototyping
  - Write pipeline from small tasks
- Reproducibility
  - Singularity / Docker containers
- Portable
  - Can run locally, with various job schedulers and, on the clouds
- Unified parallelism
  - Nextflow is based on the dataflow programming model which greatly simplifies writing complex distributed pipelines
  - Parallelization is implicitly defined by the processes input and output declarations.
- Continuous checkpoints
  - All the intermediate results produced during the pipeline execution are automatically tracked.

Features of Nextflow

- Easy to install
  - one-command install
  - can also be installed by Spack or conda
- Well documented
  - tons of documents and examples for beginners and developers at https://www.nextflow.io/docs/latest/index.html
- Strong user support
  - responsive reply from Nextflow developers on its official forum
- Java/Groove?
- Free to use!
  - open source, GPL license
  - No user/geo restrictions

What is Nextflow

- Nextflow forum: https://groups.google.com/g/nextflow
- Nextflow blogs: https://www.nextflow.io/blog.html

LSU

Installing Nextflow

- Prerequisites
  - Java 8 or later
  - Bash 3.2 or later
- On a RHEL6 cluster
  - Nextflow 20.10.0 was installed by
    ```
    curl -s https://get.nextflow.io | bash
    ```
  - Module key was manually created: `nextflow/20.10.0`
- On a RHEL7 cluster
  - Nextflow 20.10.0 was installed by Spack
  - Module key was created by Spack

Nextflow test run

- Command format: `nextflow [options] COMMAND [arg...]`

```
$ nextflow run hello
N E X T F L O W  ~  version 20.10.0
Launching `nextflow-io/hello` [modest_wilson] - revision: e6d9427e5b [master]
executor >  local (4)
[e2/802269] process > sayHello (1) [100%] 4 of 4 ✓
Ciao world!

Hola world!

Hello world!

Bonjour world!
```

Executor: where a pipeline process is run (And how many times)

hexadecimal numbers: identify the unique process execution

process name

how many times the process executed

Nextflow test run

- Other outputs

```
$ ls -a
.   ..   .nextflow   .nextflow.log   work
```

- $PWD/work
  - work directory
  - saves the cached pipeline results
  - can take of lot of disk space
- .nextflow.log
  - nextflow log file, by default at $PWD
- .nextflow
  - Nextflow system cache directory

Nextflow test run

- Resume run

```
$ nextflow run hello -resume
N E X T F L O W  ~  version 20.10.0
Launching `nextflow-io/hello` [modest_wilson] - revision: e6d9427e5b [master]
executor >  local (4)
[e2/802269] process > sayHello (1) [100%] 4 of 4 cached: 4 ✓
Ciao world!

Hola world!

Hello world!

Bonjour world!
```

Outline

- Nextflow basics
  - What is Nextflow
  - Why Nextflow
  - How to install and test Nextflow
- Nextflow process
  - Process script
  - Input qualifier
  - Output qualifier
  - Optional components “when” and “directive”
- Nextflow channel
  - channel type and factory
  - channel operator
- Nextflow executor, configuration and log
- Nextflow with Github
- Nextflow with Singularity

Nextflow first script

- Get scripts from Github

```
$ git clone https://github.com/chenyuetian/nextflow
$ cd nextflow/hello
$ nextflow run hello_v1.nf
N E X T F L O W  ~  version 20.10.0
Launching `hello_v1.nf` [grave_crick] - revision: dcaf9328c5
executor >  local (2)
[54/629fe9] process > splitLetters [100%] 1 of 1 ✓
[cf/b7d741] process > combine      [100%] 1 of 1 ✓
Hello
GACRC
```

A glance at hello_v1.nf (1)

shebang
channel as input
comments
// for single line
/*.. */ multiple
process block 1

```
#!/usr/bin/env nextflow
str = channel.value('Hello Georgia Advanced Computing Resource Center')
/*
* extract first letter of each word in str except for 'Hello'
*/
process splitLetters {
    input:
    stdin str
    output:
    path 'hello.out' into records
    """
    cat - | awk '{for (i = 2;i<=6; i++ )print \$i}'  | cut -c1  > hello.out
    """
}
```

A glance at hello_v1.nf (2)

process block 2

print output

```
/*
* combine letters
*/
process combine {
    input:
    stdin str
    path 'y' from records
    output:
    stdout into result
    """
    cat - | awk '{print \$1}'
    cat 'y' | paste -sd ''
    """
}
result.view { it.trim() }
```

Process

- A Nextflow process is the basic processing unit to execute a user script.
  - A process may contain five definition blocks, respectively: directives, inputs, outputs, when clause and finally the process script:

```
process < name > {

   [ directives ]

   input:
    < process inputs >

   output:
    < process outputs >

   when:
    < condition >

   [script|shell|exec]:
   < user script to be executed >

}
```

process 1 at hello_v1.nf

process definition and name

process input

process output

process script

process 1

```
process splitLetters {
    input:
    stdin str
    output:
    path 'hello.out' into records
    """
    cat - | awk '{for (i = 2;i<=6; i++ )print \$i}'  | cut -c1  > hello.out
    """
}
```

Process input in hello_v1 is pre-defined with a Nextflow channel before process 1:

```
str = channel.value('Hello Georgia Advanced Computing Resource Center')
```

String as process input

hello_v2.nf

```
params.str = 'Hello Georgia Advanced Computing Resource Center'
process splitLetters {
/*
 *   input:
*/
    output:
    path 'hello.out' into records

    """
    printf '${params.str}' | awk '{for (i = 2;i<=6; i++ )print \$i}'  | cut -c1
> hello.out
    """
}
```

params:
implicit variables workflow parameters specifying
in the configuration file or as command line options.

File as process input

hello_v3.nf

```
params.in = "$baseDir/hello.txt"'
process splitLetters {
    input:
    path 'x' from params.in
    output:
    path 'hello.out' into records

    '''
    count=`wc 'x' | awk '{print $2}'`
    awk -v awkcount="$count" '{for (i = 2;i<=awkcount; i++ )print $i}' < 'x' |
cut -c1  > hello.out
    '''

}
```

Applied variable "params" to support command line options:

```
$ nextflow run hello_v3.nf --in "$PWD/lsu.txt"
```

single-quote vs double-quote

hello_v3.nf

```
process splitLetters {
…
    '''
    count=`wc 'x' | awk '{print $2}'`
    awk -v awkcount="$count" '{for (i = 2;i<=awkcount; i++ )print $i}'
< 'x' | cut -c1  > hello.out
    '''
}
```

- Strings can be defined by using a single-quote or a double-quote, and multi-lines are defined by three single-quote or three double-quote characters.
- Environment variables can be referenced directly in the single-quote process script
- Not be able to use the variables defined in the pipeline script context

single-quote vs double-quote

hello_v4.nf

```
process splitLetters {
…
    """
    count=`wc 'x' | awk '{print \$2}'`
    awk -v awkcount="\$count" '{for (i = 2;i<=awkcount; i++ )print
\$i}' < 'x' | cut -c1  > hello.out
    """
}
```

- Dollar character ($) in the double-quote string script process is interpreted as a Nextflow variable placeholder
- Escape the environment variables by \

Conditional scripts

hello_v5.nf

```
params.str = "initial"
process splitLetters {
…
    script:
    if( params.str == 'initial' )
        """
        count=`wc 'x' | awk '{print \$2}'`
        awk -v awkcount="\$count" '{for (i = 2;i<=awkcount; i++ )print
\$i}' < 'x' | cut -c1  > hello.out
        """
    else if (params.str == 'full' )
        """
        count=`wc 'x' | awk '{print \$2}'`
        awk -v awkcount="\$count" '{for (i = 2;i<=awkcount; i++ )print
\$i " "}' < 'x'  > hello.out
        """
    else
        error "Invalid alignment mode: ${params.str}"}
```

"script:" is required

Template files in process script

hello_v6.nf

```
process splitLetters {
…
     script:
    template 'split.sh'
}
```

“script:” is required

- Template files can be reused across different processes and tested independently from the overall pipeline execution.

```
$ cat split.sh
#!/bin/bash
count=`wc $x | awk '{print $2}'`
echo $count
awk -v awkcount="$count" '{for (i = 2;i<=awkcount; i++ )print $i}' < $x | cut
-c1  > hello.out
```

Process input

- Command format:

```
input:
  <input qualifier> <input name> [from <source channel>] [attributes]
```

- An input definition starts with an `<input qualifier>` and the `<input name>` , followed by the keyword `from` and the actual channel over which inputs are received. Finally some input optional attributes can be specified.
- The input qualifier declares the type of data to be received. Today will cover:
  - path: handle the received value as a path
    introduced by Nextflow version 19.10.0 and it's a drop-in replacement for the file qualifier
  - stdin: forward the received value to the process *stdin* special file.
  - env: use the received value to set an environment variable
  - each: execute the process for each entry in the input collection

Path input qualifier

- The input path is global declared first, then in the path input qualifier of the process:

```
params.in = "$baseDir/hello.txt"
process splitLetters {
input:
path 'input.fa' from params.in
```

- or directly in the process script:

```
input:
path 'input.fa' from "$baseDir/hello.txt"
```

- Provided input value should represent an absolute path location

Path input qualifier

swiss.nf

- derived from https://www.nextflow.io/example3.html but uses swissprot db
  - **create swissprot db first with databases.sh (blast required)**

```
process blast {
    input:
    path 'query.fa' from fasta_ch
    path db from db_dir
```

- The single quote on query.fa ('query.fa')
  - File name fixed, doesn't have to change along with the actual provided file
  - File is staged under the related process cache directory

```
$ ls -ltr work/03/728ac8d39914e93f08958ea2697bc8/
lrwxrwxrwx 1 ychen64 Admins    44 May 11 21:23 swissprot ->
/worka/work/ychen64/nextflow/blast/swissprot
lrwxrwxrwx 1 ychen64 Admins    85 May 11 21:23 query.fa ->
/worka/work/ychen64/nextflow/blast/work/b7/25c990c93c15f42d693627c4bdbeb6/sample.1.fa
```

"db" various based on the execution environment

LSU

Path input qualifier

swiss.nf

```
process blast {
    process blast {
    """
    blastp -db $db/$db_name -query query.fa -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}
```

- query.fa in process script
  - No need to use Dollar character ($) to reference

Stdin input qualifier

swiss_v2.nf

```
process blast {
     input:
    stdin fasta_ch
    path db from db_dir
    output:
    file 'top_hits' into hits_ch
    """
    blastp -db $db/$db_name -query - -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}
```

- The stdin input qualifier forwards the value received from a channel to the standard input of the command executed by the process.

Env input qualifier

swiss_v3.nf

```
process blast {
     input:
    path 'query.fa' from fasta_ch
    env db from db_dir
    output:
    file 'top_hits' into hits_ch
    """
    blastp -db \$db/$db_name -query query.fa -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}
```

- The env qualifier defines an environment variable in the process execution context based on the value received from the channel
- Escape the environment variables by \

Each input qualifier

swiss_v4.nf

```
params.db = ["$baseDir/swissprot/swissprot","$baseDir/landmark/landmark"]
process blast {
     process blast {
    input:
    path 'query.fa' from fasta_ch
    each db from params.db
…
…
}
```

- The each qualifier repeats the execution of a process for each item in a collection

```
$ nextflow run swiss_v4.nf
N E X T F L O W  ~  version 20.10.0
Launching `swiss_v4.nf` [hungry_noether] - revision: 1e00a41cfe
executor >  local (6)
[b5/345c5e] process > blast (2)   [100%] 2 of 2 ✓
[e0/f374dd] process > extract (4) [100%] 4 of 4✓
```

Process output

- Command format:

```
output:
  <output qualifier> <output name> [into <target
channel>[,channel,..]] [attribute [,..]]
```

- Output definitions start by an `<output qualifier>` and the `<output name>` , followed by the keyword `into` and one or more channels over which outputs are sent. Finally some optional attributes can be specified.

- The output qualifier declares the type of data to send out. Today will cover:
  - path: handle the output value as a path
    introduced by Nextflow version 19.10.0 and it's a drop-in replacement for the file qualifier
  - stdout: forward the output value to the process *stdin* special file.
  - multiple output files: execute the process for each entry in the input collection

Path output qualifier

- The path output qualifier in hello.nf examples

```
path 'hello.out' into records
```

- The main advantage of path over the file qualifier is that it allows the specification of a number of outputs to fine-control the output files.
  - details at https://www.nextflow.io/docs/latest/process.html#output-path
- The path qualifier should be preferred over file to handle process output files when using Nextflow 19.10.0 or later.

Stdout output qualifier

- The stdout output qualifier in hello.nf examples

```
stdout into result
```

Multiple output

hello_v7.nf

```
process splitLetters {
/*
 *   input:
*/
    output:
    path 'hello_*' into records mode flatten
    """
    printf "${params.str}" | awk '{for (i = 2;i<=6; i++ )print \$i}'  | cut -c1
| split -l 1 - hello_
    """
}
```

- By default all the files matching the specified glob pattern are emitted by the channel as a sole (list) item.
- Emit each file as a sole item by adding the `mode flatten` attribute in the output file declaration.
- Multiple files with prefix “hello_” will be created and go to next process

Multiple output

hello_v8.nf

```
process splitLetters {
/*
 *   input:
*/
    output:
    path 'hello_*' into records
    """
    printf "${params.str}" | awk '{for (i = 2;i<=6; i++ )print \$i}'  | cut -c1
| split -l 1 - hello_
    """
}
```

- The option “mode” is deprecated as of version 19.10.0. Use the operator “flatten” to change downstream process instead

Multiple output

hello_v8.nf

```
process combine {
   input:
    path 'y' from records.flatten()
    output:
    stdout into result
    """
    printf '${params.str}' | awk '{print \$1}'
    cat 'y'
    """
}
```

Process when

swiss_v5.nf

```
process blast {
    input:
    path 'query.fa' from fasta_ch
    path db from db_dir
    when:
    db.name =~ 'swissprot'
    output:
    file 'top_hits' into hits_ch
    """
    blastp -db $db/$db_name -query query.fa -outfmt 6 > blast_result
    cat blast_result | head -n 10 | cut -f 2 > top_hits
    """
}
```

- The “when” declaration enable/disable a process execution depending on the state of various inputs and parameters.

Process directive

- Directive provides optional settings that will affect the execution of the current process.
- They must be entered at the top of the process body, and has the syntax:

```
name value [, value2 [,..]]
```

- Some directives are available to all processes. Today will cover:
  - echo: print out stdout produced by the commands executed in process
  - module: load Environment Module in process
  - conda: create or activate conda virtual environment

Complete list available at:
https://www.nextflow.io/docs/latest/process.html#directives

Process directive

hello_v9.nf

```
process blast {
    echo true
    input:
    path 'x' from params.in
    output:
    path 'hello.out' into records
    '''
    count=`wc 'x' | awk '{print $2}'`
    echo "count is $count"
    awk -v awkcount="$count" '{for (i = 2;i<=awkcount; i++ )print $i}' < 'x' |
cut -c1  > hello.out
    '''
}
```

- echo directive is an excellent tool for testing and debugging.

Process directive

swiss_v6.nf

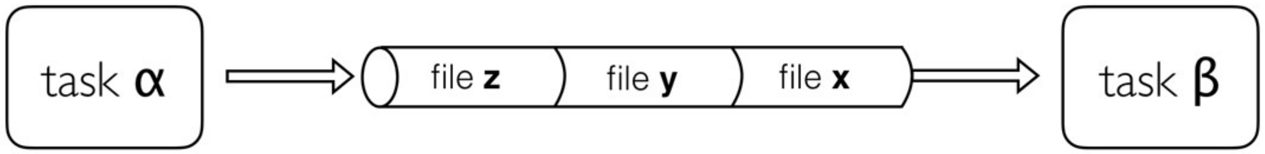
```
process blast {
    module 'python'
    conda '/project/ychen64/conda/envs/blast-2.11.0 '
…
…
}
```

- Python module is loaded for using conda
- conda environment is activated.

Outline

Channel

- Nextflow processes communicate through channels.
  - Sending a message is an *asynchronous* operation which completes immediately, without having to wait for the receiving process.
  - Receiving data is a blocking operation which stops the receiving process until the message has arrived.



https://nextflow-io.github.io/nf-hack18/training.html#_channels

Channel types

- Queue channel: A queue channel is a non-blocking unidirectional FIFO queue which connects two processes or operators.
  - created by a factory method such as a `from`, `fromPath`, etc. or chaining it with a channel operator such as `map`, `flatMap`, etc.
  - Queue channels are also created by process output declarations using the into clause.

```
process foo {
  echo true
  input:
  val x from Channel.from(1,2)
  val y from Channel.from('a','b','c')
  script:
   """
   echo $x and $y
   """
}
```

Output:

1 and a

2 and b

Channel types

- Value channel: Bound to a single value and it can be read unlimited times without consuming its content.
  - created by a factory method `value`
  - or by operators returning a single value, such as `first`, `last`, `collect`, `count`, `min`, `max`, `reduce`, `sum`

```
process bar {
  echo true
  input:
  val x from Channel.value(1)
  val y from Channel.from('a','b','c')
  script:
   """
   echo $x and $y
   """
}
```

Output:

```
1 and a
1 and b
1 and c
```

Channel factory

- Channels may be created explicitly.

Some important channel factory methods

value	used to create a value channel. For example: expl2 = Channel.value('Hello there')
of	create a channel emitting any sequence of values that are specified as the method argument, for example: ch = Channel.of(1, 3, 5, 7)
fromList	create a channel emitting the values provided as a list of elements
fromPath	create a channel emitting one or more file paths by using the fromPath method and specifying a path string as an argument
fromFilePairs	create a channel emitting the file pairs matching a glob pattern provided by the user
empty	create a channel that doesn't emit any value
create	deprecated
from	deprecated and should only be used for backward compatibility in legacy code. Use of or fromList instead.

Operator

- Connect channels to each other or to transform values emitted by a channel applying some user provided rules. Full list is at:

  https://www.nextflow.io/docs/latest/operator.html#
- Commonly used operators:
  - view: prints the items emitted by a channel to the console standard output.

    swiss.nf:
    ```
    .view { file -> "matching sequences:\n ${file.text}" }
    ```
  - collectFile: gathers the items emitted by a channel and save them to file(s)

    swiss.nf:
    ```
    .collectFile(name: params.out)
    ```

Outline

- Nextflow basics
  - What is Nextflow
  - Why Nextflow
  - How to install and test Nextflow
- Nextflow process
  - Process script
  - Input qualifier
  - Output qualifier
  - Optional components “when” and “directive”
- Nextflow channel
  - channel type and factory
  - channel operator
- Nextflow executor, configuration and log
- Nextflow with Github
- Nextflow with Singularity

Executor

- The executor is the component that determines the system where a pipeline process is run and supervises its execution.
- Local: The local executor is used by default.

```
$ nextflow run hello
N E X T F L O W  ~  version 20.10.0
Launching `nextflow-io/hello` [modest_wilson] -
revision: e6d9427e5b [master]
executor >  local (4)
```

Executor

- The executor is the component that determines the system where a pipeline process is run and supervises its execution.
- PBS: run your pipeline script by using PBS/Torque scheduler
  - set the property `process.executor = 'pbs'` in the `nextflow.config` file

```
$ cat nextflow.config
executor {
    name = 'pbs'
}
```

  - define in EVERY process directive

hello_v10.nf

```
process blast {
    queue = 'checkpt'
    time = '30m'
    clusterOptions = '-A hpc_hpcadmin7 -l nodes=1:ppn=20'
```

Each process will start a job

Executor

- The executor is the component that determines the system where a pipeline process is run and supervises its execution.
- SLURM: run your pipeline script by using SLURM scheduler
  - set the property `process.executor = 'slurm'` in the `nextflow.config` file

```
$ cat nextflow.config
executor {
    name = 'slurm'
}
```

  - define in EVERY process directive

hello_v11.nf

```
process blast {
    queue = 'checkpt'
    time = '30m'
    clusterOptions = '-A hpc_hpcadmin7 --N1'
```

Each process will start a job

Configuration

- Configuration files can be placed in multiple locations
- Possible configuration sources listed in order of priority
  1. Parameters specified on the command line (`--something 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
  6. The config file `$HOME/.nextflow/config`
  7. Values defined within the pipeline script itself (e.g. `main.nf`)
- Config syntax

```
name = value
```

- Ignore any default configuration files and use only the custom one use the command line option `-C <config file>`

Configuration applications

- `nextflow.config` in the current directory to select executor
- `nextflowVersion` setting allows you to specify a minimum required version to run the pipeline.

```
$ cat nextflow.config
manifest{
nextflowVersion = '>=21.04.0'
}
$ nextflow run hello_v9.nf
N E X T F L O W  ~  version 20.10.0
Launching `hello_v9.nf` [exotic_mercator] - revision: b171969dd3
WARN: Nextflow version 20.10.0 does not match workflow required version:
>=21.04.0 -- Execution will continue, but things may break!
```

Nextflow log

- Showing information about executed pipelines in the **current** folder

```
$ nextflow log
  TIMESTAMP              DURATION       RUN NAME               STATUS   REVISION ID     SESSION ID
  COMMAND
  2021-05-05 13:05:55    2.2s           chaotic_venter         OK       e6d9427e5b      276a9672-85a2-4b9a-
  9fba-67f7a5c83c9a    nextflow run hello
  2021-05-05 13:07:39    4.1s           magical_mandelbrot     OK       b0aa2a5dd9      ca70b869-31e9-4d6f-
  88b4-d8a17bf8d891    nextflow run blast.nf
  nextflow run blast_swiss.nf
```

- Specifying a run name or session id prints tasks :

```
 $ nextflow log chaotic_venter
/worka/project/ychen64/test/blast-2.10.1/work/62/4538246bf7413109db1ce7c9b55fb4
/worka/project/ychen64/test/blast-2.10.1/work/cb/62794ad073387838b937c22e270aa8
```

- Customizing fields

```
 $ nextflow log chaotic_venter -f hash,status
62/453824       COMPLETED
cb/62794a       COMPLETED
.
```

Outline

- Nextflow basics
  - What is Nextflow
  - Why Nextflow
  - How to install and test Nextflow
- Nextflow process
  - Process script
  - Input qualifier
  - Output qualifier
  - Optional components "when" and "directive"
- Nextflow channel
  - channel type and factory
  - channel operator
- Nextflow executor, configuration and log
- Nextflow with Github
- Nextflow with Singularity

Pipeline sharing with Github

- When `main.nf` exists at `http://github.com/foo/bar`
- NextFlow will pull github repo to `$HOME/.Nextflow/asset`

```
$ nextflow run chenyuetian/nextflow
Pulling chenyuetian/nextflow ...
downloaded from https://github.com/chenyuetian/nextflow.git
Project `chenyuetian/nextflow` currently is sticked on revision: main -- you
need to specify explicitly a revision with the option `-r` to use it
```

- `main.nf` can be empty, the whole repo will be downloaded anyway
- set `nextflow.configure` file **on the remote repo** if another nf name is preferred:

```
manifest {
mainScript = 'bash.nf'
}
```

Pipeline sharing with Github

- Listing available projects (in the local repo)

```
$ nextflow list
chenyuetian/nextflow
nextflow-io/hello
```

- Showing project information (in the local repo)

```
$ nextflow info chenyuetian/nextflow
 project name: chenyuetian/nextflow
 repository  : https://github.com/chenyuetian/nextflow
 local path  : /home/ychen64/.nextflow/assets/chenyuetian/nextflow
 main script : bash.nf
 description : on github
 revision    : * main
```

- Viewing the project code

```
$ nextflow view chenyuetian/nextflow
== content of file: /home/ychen64/.nextflow/assets/chenyuetian/nextflow/bash.nf
```

Pipeline sharing with Github

- Pulling or updating a project

```
$ nextflow pull nextflow-io/hello
```

- Cloning a project into a folder

```
$ nextflow clone nextflow-io/hello target-dir
```

- Deleting a downloaded project

```
$ nextflow drop nextflow-io/hello
```

Other commands of Nextflow

- Clean up cache and work directories.
  - A list of of run names and session ids can be generated by invoking `nextflow log -q`

```
nextflow clean [run_name|session_id] [options]
```

Name, shorthand (if any)	Default	Description
-after		Clean up runs executed *after* the specified one.
-before		Clean up runs executed *before* the specified one.
-but		Clean up all runs *except* the specified one.
-dry-run, -n	false	Print names of files to be removed without deleting them.
-force, -f	false	Force clean command.
-help, -h	false	Print the command usage.
-keep-logs, -k	false	Removes only temporary files but retains execution log entries and metadata.
-quiet, -q	false	Do not print names of files removed.

- Print the resolved pipeline configuration.

```
$ nextflow config
manifest {
   nextflowVersion = '>=19.10.0'
}
```

Outline

Nextflow run with Singularity

- Prerequisites
  - singularity on execution environment
  - user defined bind points if not run in /home

  recipe: https://github.com/chenyuetian/nextflow/singularity/singularity.nextflow
- Various tools including Nextflow itself is inside the container

```
$ module purge # unload any Nextflow modules outside
$ singularity exec -B /project,/work nextflow.simg nextflow run swiss_v7.nf
N E X T F L O W  ~  version 20.10.0
Launching `swiss_v7.nf` [stoic_mayer] - revision: 15e3a4cfc3
…
…
```

Nextflow run with Singularity

swiss_v7.nf

```
params.db = "/usr/local/swissprot/swissprot“
db_dir = file(params.db).parent
process blast {
    input:
    path 'query.fa' from fasta_ch
    env db from db_dir
```

- The database `swissprot` is included in the container, totally portable
- Use env input qualifier so there won’t be missing file/link in the cache

Nextflow run with Singularity

- Using Nextflow's built-in support for Singularity (Nextflow must be available outside of the container)

```
  $ nextflow run  swiss_v7.nf -with-singularity nextflow.simg
N E X T F L O W  ~  version 20.10.0
Launching `swiss_v7.nf` [cheesy_gilbert] - revision: ab0bc59e35
executor >  local (2)
[be/c67f31] process > blast (1)   [100%] 1 of 1 ✓
[78/32f835] process > extract (1) [100%] 1 of 1 ✓
```

- If command above, binding control is required for runs not in /home:

```
$ cat nextflow.config
singularity {
    autoMounts = true
}
```

Not Covered

- Groovy scripting
  - language basics
  - Implicit variables
  - Files and I/O
- Scripts à la carte: mix scripting language (e.g. Perl, Python, Ruby, R, etc) in the same pipeline
- Almost all channel operators
- DSL 2
- Reporting & visualization
- Workflow introspection
- Mail & Notifications

Questions?