

Introduction to automation using Nextflow:

A tutorial through examples

Phelelani Mpangase

28 September 2020

Sydney Brenner Institute for Molecular Bioscience
University of the Witwatersrand
Johannesburg
South Africa

Outline

- 1 Introduction to Nextflow
 - Introduction
 - Nextflow Script
 - Partial Execution
 - Visualising the Workflow
- 2 Groovy
 - Groovy Closures
- 3 Generalising and Extending
 - Parameters
 - Channels
 - Generalising Our Example
 - Managing Grouped Files
 - On absolute paths
- 4 Nextflow and Docker
 - Docker & Singularity Containers
 - Directory & File Access
- 5 Executors
 - Executors
 - Nextflow on a cluster (HPC)
 - Scheduler + Docker
 - Amazon EC2
- 6 Channel Operations
 - Using **join**
 - Using **merge**
 - **join** vs **merge**
 - Working version of the example
 - Copying channels

Introduction to Nextflow

Introduction to Nextflow

Introduction

Resources

- <https://github.com/phelelani/nf-tut-2020>

Workflow Languages

Many scientific applications require

- Multiple data files
- Multiple applications
- Perhaps different parameters

General purpose languages not well suited

- Too low a level of abstraction
- Does not separate workflow from application
- Not reproducible

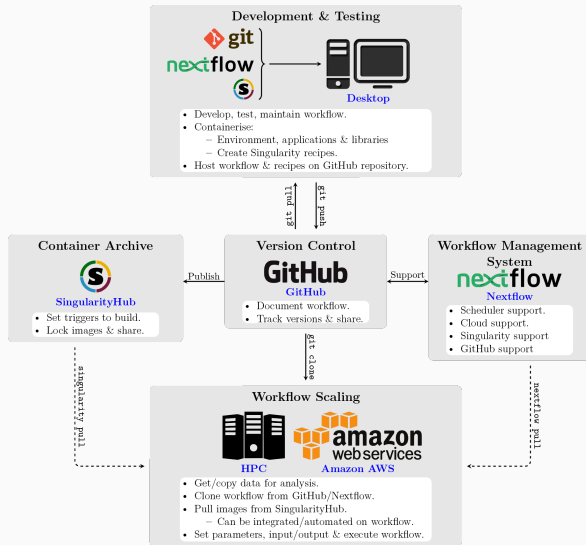
Workflow Languages

Many scientific applications require

- Multiple data files
- Multiple applications
- Perhaps different parameters

General purpose languages not well suited

- Too low a level of abstraction
- Does not separate workflow from application
- Not reproducible



Nextflow

Groovy-based language

- Expressing workflows
- Portable
 - works on most Unix-like systems
- Very easy to install
 - NB: requires Java 7, 8
- Scalable
- Supports Docker/Singularity
- Supports a range of scheduling systems

Nextflow

Groovy-based language

- Expressing workflows
- Portable
 - works on most Unix-like systems
- Very easy to install
 - NB: requires Java 7, 8
- Scalable
- Supports Docker/Singularity
- Supports a range of scheduling systems

Key concepts of Nextflow

- **Processes:**
 - actual work being done (usually simple).
 - call program that does the analysis.
- **Channels:**
 - for communication between processes.
 - handles inputs and outputs.
- When all inputs ready, process is executed.
- Each process runs in its own directory (files are staged).
- Supports resumption of previous partial runs.

Introduction to Nextflow

Nextflow Script

Simple Example: Using BASH

Input is a file

- With 6 columns
- Column 2 is an index column
- Identify rows with identical field 2
- Remove identical rows

```
11  11:189256  0  189256  A  G
11  11:193788  0  193788  T  C
11  11:194062  0  194062  T  C
11  11:194228  0  194228  A  G
11  11:193788  0  193788  A  C
```

Using BASH:

```
cut -f 2 data/11.bim | sort | uniq -d > dups
grep -v -f dups data/11.bim > 11.clean
```

Simple Example: Using nextflow

```
1  #!/usr/bin/env nextflow
2
3  input_ch = Channel.fromPath("data/11.bim")
4
5  process getIDs {
6      input:
7          file input from input_ch
8
9      output:
10         file "ids" into id_ch
11         file "11.bim" into orig_ch
12
13     script:
14         "cut -f 2 $input | sort > ids"
15 }
16
17 process getDups {
18     input:
19         file input from id_ch
20
21     output:
22         file "dups" into dups_ch
23
24     script:
25         """
26         uniq -d $input > dups
27         touch ignore
28         """
29 }
```

Simple Example: Using nextflow

```

1  #!/usr/bin/env nextflow
2
3  input_ch = Channel.fromPath("data/11.bim")
4
5  process getIDs {
6      input:
7          file input from input_ch
8
9      output:
10         file "ids" into id_ch
11         file "11.bim" into orig_ch
12
13     script:
14         "cut -f 2 $input | sort > ids"
15 }
16
17 process getDups {
18     input:
19         file input from id_ch
20
21     output:
22         file "dups" into dups_ch
23
24     script:
25         """
26         uniq -d $input > dups
27         touch ignore
28         """
29 }

```

```

30 process removeDups {
31     input:
32         file badids from dups_ch
33         file orig from orig_ch
34
35     output:
36         file "clean.bim" into output
37
38     script:
39         "grep -v -f $badids $orig > clean.bim "
40 }
41
42 output.subscribe { print "Done!" }

```

Simple Example: Using nextflow

```

1  #!/usr/bin/env nextflow
2
3  input_ch = Channel.fromPath("data/11.bim")
4
5  process getIDs {
6      input:
7          file input from input_ch
8
9      output:
10         file "ids" into id_ch
11         file "11.bim" into orig_ch
12
13     script:
14         "cut -f 2 $input | sort > ids"
15 }
16
17 process getDups {
18     input:
19         file input from id_ch
20
21     output:
22         file "dups" into dups_ch
23
24     script:
25         """
26         uniq -d $input > dups
27         touch ignore
28         """
29 }

```

```

30 process removeDups {
31     input:
32         file badids from dups_ch
33         file orig from orig_ch
34
35     output:
36         file "clean.bim" into output
37
38     script:
39         "grep -v -f $badids $orig > clean.bim "
40 }
41
42 output.subscribe { print "Done!" }

```

```
$ nextflow run cleandups.nf
```

```

N E X T F L O W ~ version 19.04.1
Launching `cleandups.nf` [soggy_jennings] - revision: 795e2aa39d
[warm up] executor > local
executor > local (3)
[84/7e1ad1] process > getIDs      [100%] 1 of 1 ▢
[19/cc8bf9] process > getDups     [100%] 1 of 1 ▢
[f9/ed086d] process > removeDups [100%] 1 of 1 ▢
Completed at: 31-Jul-2019 09:00:50
Duration    : 1.5s
CPU hours   : (a few seconds)
Succeeded   : 3

```

Simple Example: Using nextflow

The work directory

```
--work
| |--90
| | |--cebf3649d883f88381e32b4912b560
| | | |--ids -> /Users/phele/day4/work/b3/aa0380f2a1bca447259b7ffd390083/ids
| | | |--ignore
| |--9c
| | |--e0cb7d8d26682d7d4a1c44392f2bb3
| | | |--11.bim -> /Users/phele/day4/data/11.bim
| | | |--clean.bim
| | | |--dups -> /Users/phele/day4/work/90/cebf3649d883f88381e32b4912b560/dups
| |--b3
| | |--aa0380f2a1bca447259b7ffd390083
| | | |--11.bim -> /Users/phele/day4/data/11.bim
| | | |--ids
```

Introduction to Nextflow

Partial Execution

Partial Execution

If execution of workflow is only partial

- Because of error
- Only need to resume from process that failed

```
nextflow run cleandups.nf -resume
```

Introduction to Nextflow

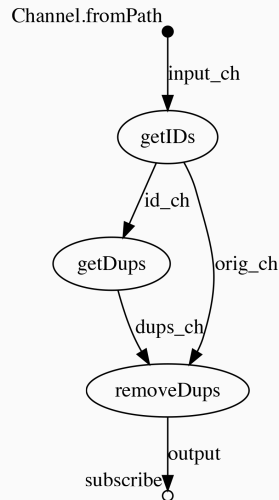
Visualising the Workflow

Visualising the Workflow

Nextflow supports several visualisation tools:

-with-dag

```
nextflow run cleandups.nf -with-dag <file-name>
```



Visualising the Workflow

Nextflow supports several visualisation tools:

-with-dag

```
nextflow run cleandups.nf -with-dag <file-name>
```

-with-timeline

```
nextflow run cleandups.nf -with-timeline <file-name>
```

Processes execution timeline

Launch time: 05 Jun 2018 10:41
Elapsed time: 2.9s



Created with Nextflow -- <http://nextflow.io>

Visualising the Workflow

Nextflow supports several visualisation tools:

-with-dag

```
nextflow run cleandups.nf -with-dag <file-name>
```

-with-timeline

```
nextflow run cleandups.nf -with-timeline <file-name>
```

-with-report

```
nextflow run cleandups.nf -with-report <filename>
```

The screenshot displays a 'Nextflow Report' for a workflow named 'cleandups.nf'. The report is titled 'Nextflow workflow report' and is generated by 'evil_keller'. It indicates that the 'Workflow execution completed successfully!'. The 'Run times' section shows the workflow was executed on 'Tue Jun 05 10:41:20 SAST 2018' and completed 5 minutes ago with a duration of 1.8s. A green progress bar shows '3 succeeded'. The 'Nextflow command' used is 'nextflow run cleandups.nf -with-timeline timeline.html -with-dag timeline.dot -with-report report.html'. A table at the bottom lists various metadata items:

CPU-Hours	{a few seconds}
Launch directory	/home/phelani/2018_courses/nextflow-course
Work directory	/home/phelani/2018_courses/nextflow-course/work
Project directory	/home/phelani/2018_courses/nextflow-course
Script name	cleandups.nf
Script ID	795e2aa39d5c85a2d0961c9f8486c1e
Workflow session	eb918148-7943-4369-bca5-7cc8e5d294e3
Workflow profile	standard
Nextflow version	version 0.29.1, build 4804 (10-05-2018 07:47 UTC)

Groovy

Groovy

Nextflow is a DSL built with Groovy

- Can inter-mix Nextflow, Groovy and Java code.
- Very powerful, flexible.
- Don't need to know much (any?) Groovy but a little knowledge is a powerful thing

Groovy

Groovy Closures

Groovy: Closures

Closures are anonymous functions

- Similar to lambdas in Python
- Don't want the overhead of naming a function we only use once
- Typically use with higher-order functions
 - Functions that take other functions as arguments
- Very powerful and useful

Syntax for a closure that takes one argument:

```
{ parm -> expression }
```

Groovy: Closures

Closures are anonymous functions

- Similar to lambdas in Python
- Don't want the overhead of naming a function we only use once
- Typically use with higher-order functions
 - Functions that take other functions as arguments
- Very powerful and useful

Syntax for a closure that takes one argument:

```
{ parm -> expression }
```

```
1 { a -> a*a } (3)
2
3 { a -> a*a+7*a - 2 } (3)
4
5 for (n in 1..5) print( {it*it} (n));
6
7 { x, y -> Math.sqrt(x*x + y*y) } (3,4)
8
9 int doX(f, nums) {
10     sum=0;
11     for ( n in nums ) {
12         sum = sum+f(n);
13     }
14     return sum
15 }
16
17 print doX ( {a->a}, [4,5,16] );
18
19 print doX ( {a->a*a}, [4,5,16] );
20
21 print doX ( { it*it }, [4,5,16]);
22
23 m=10
24
25 print doX({a->m*a+2}, [1,2,3])
```

Generalising and Extending

Extending the Example

- Parameterise the input
- Want output to go to convenient place
- Workflow takes in multiple input files – processes are executed on each in turn.
- Complication : may need to carry the base name of the input to the final output;
- Can repeat some steps for different parameters.

Generalising and Extending

Parameters

Parameters

In Nextflow file:

```
input_ch = Channel.fromPath(params.data_dir)
```

And run it like this

```
nextflow run phylo1.nf --data_dir data/polyseqs.fa
```

Generalising and Extending

Channels

Data Types in Channels

Channels support different types:

- file
- val
- set

Creating Channels

Many, many operations you can do on channels and their contents

bind	buffer	close
filter	map/reduce	group
join, merge	mix	copy
split	spread	fork
count	min/max/sum	print/view

```
Channel.create()
Channel.empty
Channel.from("blast","plink")
Channel.fromPath("data/*.fa")
Channel.fromFilePairs("data/{YRI,CEU,BEB}.*")
Channel.watchPath("*fa")
```


Generalising and Extending

Generalising Our Example

Workflow: Multiple Inputs

```
1  params.data_dir = "data"
2  input_ch = Channel.fromPath("${params.data_dir}/*.bim")
3
4  process getIDs {
5      input:
6          file input from input_ch
7
8      output:
9          file "${input.baseName}.ids" into id_ch
10         file "$input" into orig_ch
11
12     script:
13         "cut -f 2 $input | sort > ${input.baseName}.ids"
14 }
15
16 process getDups {
17     input:
18         file input from id_ch
19
20     output:
21         file "${input.baseName}.dups" into dups_ch
22
23     script:
24         out = "${input.baseName}.dups"
25         ""
26         uniq -d $input > $out
27         touch ignore
28         ""
29 }
```

Workflow: Multiple Inputs

```
1  params.data_dir = "data"
2  input_ch = Channel.fromPath("${params.data_dir}/*.bim")
3
4  process getIDs {
5      input:
6          file input from input_ch
7
8      output:
9          file "${input.baseName}.ids" into id_ch
10         file "$input" into orig_ch
11
12     script:
13         "cut -f 2 $input | sort > ${input.baseName}.ids"
14 }
15
16 process getDups {
17     input:
18         file input from id_ch
19
20     output:
21         file "${input.baseName}.dups" into dups_ch
22
23     script:
24         out = "${input.baseName}.dups"
25         ""
26         uniq -d $input > $out
27         touch ignore
28         ""
29 }
```

```
30 process removeDups {
31     publishDir "output", pattern: "${badids.baseName}_clean.bim"
32         , overwrite:true, mode:'copy'
33
34     input:
35         file badids from dups_ch
36         file orig from orig_ch
37
38     output:
39         file "${badids.baseName}_clean.bim" into cleaned_ch
40
41     script:
42         "grep -v -f $badids $orig > ${badids.baseName}_clean.bim "
43 }
```

Workflow: Multiple Inputs

```
1  params.data_dir = "data"
2  input_ch = Channel.fromPath("${params.data_dir}/*.bim")
3
4  process getIDs {
5      input:
6          file input from input_ch
7
8      output:
9          file "${input.baseName}.ids" into id_ch
10         file "$input" into orig_ch
11
12     script:
13         "cut -f 2 $input | sort > ${input.baseName}.ids"
14 }
15
16 process getDups {
17     input:
18         file input from id_ch
19
20     output:
21         file "${input.baseName}.dups" into dups_ch
22
23     script:
24         out = "${input.baseName}.dups"
25         ""
26         uniq -d $input > $out
27         touch ignore
28         ""
29 }
```

```
30 process removeDups {
31     publishDir "output", pattern: "${badids.baseName}_clean.bim"
32         , overwrite:true, mode:'copy'
33
34     input:
35         file badids from dups_ch
36         file orig from orig_ch
37
38     output:
39         file "${badids.baseName}_clean.bim" into cleaned_ch
40
41     script:
42         "grep -v -f $badids $orig > ${badids.baseName}_clean.bim "
```

```
$ nextflow run cleandups.nf
```

```
Launching `cleandups.nf` [distracted_hodgkin] - revision: 29
fdb384a6
[warm up] executor > local
executor > local (9)
[1a/431eb7] process > getIDs [100%] 3 of 3
[cc/fc0aaa] process > getDups [100%] 3 of 3
[03/c31154] process > removeDups [100%] 3 of 3
Completed at: 31-Jul-2019 10:26:23
Duration : 2s
CPU hours : (a few seconds)
Succeeded : 9
```

Workflow: Multiple Parameters

Now try splitting the file but use different split values

```
split -l 400 data.txt dataX
```

will produce files dataXaa, dataXab, dataXac and so on ...

Try:

```
1 splits = [400,500,600]
2
3 process splitIDs {
4   input:
5     file bim from cleaned_ch
6     each split from splits
7
8   output:
9     file ("*-$split-") into output_ch;
10
11   script:
12     "split -l $split $bim ${bim.baseName}-$split- "
13 }
```

Generalising and Extending

Managing Grouped Files

Grouped Files

Use PLINK as an example.

```
## Short version of the command
plink --bfile /path/YRI --freq --out /tmp/YRI

## Long version of the command
plink --bed YRI.bed \
      --bim YRI.bim \
      --fam YRI.fam \
      --freq \
      --out /tmp/YRI
```

Problem:

- Pass the files on another channel(s) to be staged
- Pass the base name as value/or work it out

Pros/Cons

- Simple
- Need extra channel/some gymnastics

Grouped Files

Use PLINK as an example.

```
## Short version of the command
plink --bfile /path/YRI --freq --out /tmp/YRI

## Long version of the command
plink --bed YRI.bed \
  --bim YRI.bim \
  --fam YRI.fam \
  --freq \
  --out /tmp/YRI
```

Problem:

- Pass the files on another channel(s) to be staged
- Pass the base name as value/or work it out

Pros/Cons

- Simple
- Need extra channel/some gymnastics

RECAP CLOSURES

Simply, a *closure* is an anonymous function

- Code wrapped in braces {, }
- Default argument called *it*

```
[1,2,3].each { print it * it }
[1,2,3].each { num -> print num * num }
```


Grouped Files - Version 1: map

```
1  #!/usr/bin/env nextflow
2  params.dir = "data/pops/"
3  dir = params.dir
4  params.pops = ["YRI", "CEU", "BEB"]
5
6  Channel
7      .from(params.pops)
8      .map { pop ->
9          [ file("$dir/${pop}.bed"),
10            file("$dir/${pop}.bim"),
11            file("$dir/${pop}.fam")]
12      }
13      .set { plink_data }
14
15  plink_data.subscribe { println "$it" }
```

Grouped Files - Version 1: map

```
1 #!/usr/bin/env nextflow
2 params.dir = "data/pops/"
3 dir = params.dir
4 params.pops = ["YRI", "CEU", "BEB"]
5
6 Channel
7     .from(params.pops)
8     .map { pop ->
9         [ file("${dir}${pop}.bed"),
10           file("${dir}${pop}.bim"),
11           file("${dir}${pop}.fam")]
12     }
13     .set { plink_data }
14
15 plink_data.subscribe { println "$it" }
```

```
[data/pops/YRI.bed, data/pops/YRI.bim, data/pops/YRI.fam]
[data/pops/CEU.bed, data/pops/CEU.bim, data/pops/CEU.fam]
[data/pops/BEB.bed, data/pops/BEB.bim, data/pops/BEB.fam]
```

Grouped Files - Version 1: map

```
1 #!/usr/bin/env nextflow
2 params.dir = "data/pops/"
3 dir = params.dir
4 params.pops = ["YRI", "CEU", "BEB"]
5
6 Channel
7     .from(params.pops)
8     .map { pop ->
9         [ file("${dir}/${pop}.bed"),
10           file("${dir}/${pop}.bim"),
11           file("${dir}/${pop}.fam") ]
12     }
13     .set { plink_data }
14
15 plink_data.subscribe { println "$it" }
```

```
16 process getFreq {
17     input:
18         set file(bed), file(bim), file(fam) from plink_data
19     output:
20         file "${bed.baseName}.frq" into result
21
22     """
23     plink --bed $bed \
24           --bim $bim \
25           --fam $fam \
26           --freq \
27           --out ${bed.baseName}"
28     """
29 }
```

```
[data/pops/YRI.bed, data/pops/YRI.bim, data/pops/YRI.fam]
[data/pops/CEU.bed, data/pops/CEU.bim, data/pops/CEU.fam]
[data/pops/BEB.bed, data/pops/BEB.bim, data/pops/BEB.fam]
```

Grouped Files - Version 2: `fromFilePairs`

Use `fromFilePairs`.

- Takes a closure used to gather files together with the same key

```
x_ch = Channel.fromFilePairs( files ) { closure }
```

- Specify the files as a glob
- Closure associates each file with a key
- `fromPairs` puts all files with same key together
- Returns a list of pairs (key, list)

Grouped Files - Version 2: `fromFilePairs`

Use `fromFilePairs`.

- Takes a closure used to gather files together with the same key

```
x_ch = Channel.fromFilePairs( files ) { closure }
```

- Specify the files as a glob
- Closure associates each file with a key
- `fromPairs` puts all files with same key together
- Returns a list of pairs (key, list)

```
1  #!/usr/bin/env nextflow
2
3  commands = Channel.fromFilePairs("/usr/bin/*", size:-1) {
4      it.baseName[0]
5  }
6
7  commands.subscribe { k= it[0];
8      n=it[1].size();
9      println "There are $n files starting with $k";
10 }
```

A more complex example – default closure

```
1  Channel
2      .fromFilePairs
3      ("${params.dir}/*.{bed,fam,bim}",size:3, flat : true)
4      .ifEmpty { error "No matching plink files" }
5      .set { plink_data }
6
7  plink_data.subscribe { println "$it" }
```

Grouped Files - Version 2: `fromFilePairs`

Use `fromFilePairs`.

- Takes a closure used to gather files together with the same key

```
x_ch = Channel.fromFilePairs( files ) { closure }
```

- Specify the files as a glob
- Closure associates each file with a key
- `fromPairs` puts all files with same key together
- Returns a list of pairs (key, list)

```
[CEU, [data/pops/CEU.bed, data/pops/CEU.bim, data/pops/CEU.fam]]  
[YRI, [data/pops/YRI.bed, data/pops/YRI.bim, data/pops/YRI.fam]]  
[BEB, [data/pops/BEB.bed, data/pops/BEB.bim, data/pops/BEB.fam]]
```

```
1  #!/usr/bin/env nextflow  
2  
3  commands = Channel.fromFilePairs("/usr/bin/*", size:-1) {  
4      it.baseName[0]  
5  }  
6  
7  commands.subscribe { k= it[0];  
8      n=it[1].size();  
9      println "There are $n files starting with $k";  
10 }
```

A more complex example – default closure

```
1  Channel  
2      .fromFilePairs  
3      ("${params.dir}/*.{bed,fam,bim}",size:3, flat : true)  
4      .ifEmpty { error "No matching plink files" }  
5      .set { plink_data }  
6  
7  plink_data.subscribe { println "$it" }
```

Grouped Files - Version 2: fromFilePairs

```
1 process checkData {
2   input:
3     set pop, file(pl_files) from plink_data
4
5   output:
6     file "${pl_files[0]}.frq" into result
7
8   script:
9     base = pl_files[0].baseName
10    "plink --bfile $base --freq --out ${base}"
11 }
```

Grouped Files - Version 2: fromFilePairs

```
1 process checkData {  
2   input:  
3     set pop, file(pl_files) from plink_data  
4  
5   output:  
6     file "${pl_files[0]}.frq" into result  
7  
8   script:  
9     base = pl_files[0].baseName  
10    "plink --bfile $base --freq --out ${base}"  
11 }
```

```
1 process checkData {  
2   input:  
3     set pop, file(pl_files) from plink_data  
4  
5   output:  
6     file "${pop}.frq" into result  
7  
8   script:  
9     "plink --bfile $pop --freq --out $pop"  
10 }
```


Grouped Files - Final Version

```
1 #!/usr/bin/env nextflow
2
3  params.dir = "data/pops/"
4  dir = params.dir
5  params.pops = ["YRI", "CEU", "BEB"]
6
7  Channel
8    .fromFilePairs("${params.dir}/{YRI,BEB,CEU}.{bed,bim,fam}",size:3) {
9      file -> file.baseName
10    }
11    .filter { key, files -> key in params.pops }
12    .set { plink_data }
13
14  process checkData {
15    input:
16      set pop, file(pl_files) from plink_data
17
18    output:
19      file "${pop}.frq" into result
20
21    script:
22      "plink --bfile $pop --freq --out $pop"
23  }
```

Generalising and Extending

On absolute paths

Absolute paths

```
1 input = Channel.fromPath("/data/batch1/myfile.fa")
2
3 process show {
4     input:
5     file data from input
6
7     output:
8     file 'see.out'
9
10    script:
11    cp $data /home/scott/answer
12    ...
```

Nextflow and Docker

Nextflow and Docker

Docker & Singularity Containers

Docker & Singularity Containers

Light-weight virtualisation abstraction layer

- Currently runs on Unix like systems
 - Linux
 - macOS
- Windows support coming

Can create images locally or get from repositories

```
## Docker
docker pull ubuntu
docker pull quay.io/banshee1221/h3agwas-plink
```

```
## Singularity
singularity pull docker://ubuntu
singularity pull docker://quay.io/banshee1221/h3agwas-plink
```

Running images

```
## Docker
docker run <some-image-name>
```

```
## Singularity
singularity exec <some-image-name>
```

- Docker/Singularity often run images in background
- Can also run interactively

Running Docker interactively

```
sudo docker run -t -i quay.io/banshee1221/h3agwas-plink
```

Running Singularity interactively

```
singularity shell docker://quay.io/banshee1221/h3agwas-plink
```

Nextflow supports Docker & Singularity

- Well designed script should be highly portable
- Each process gets run as a separate image call
 - Under the hood, a **docker run** or a **singularity exec** is called
- Can use the same or different images for each process
 - Parameterisable

Assuming all processes use the same image:

```
## For Docker
nextflow run plink2.nf -with-docker quay.io/banshee1221/h3agwas-plink

## For Singularity
nextflow run plink.nf -with-singularity docker://quay.io/banshee1221/h3agwas-plink
```

Nextflow and Docker

Directory & File Access

Directory & File access

Nextflow Docker/Singularity support highly transparent – but pay attention to good practice

- For each process Docker/Singularity mounts the work directory for **that** process on the Docker/Singularity image.
- Files can be staged in and out using Nextflow mechanisms.
- Other files available: directories mounted through Docker/Singularity run time options or on the Docker image
- No other files on the host machine including the current directory
- Process executes in the Docker/Singularity environment

Directory & File access

```
1 data = Channel.fromPath("data/pops/YRI.bim")
2
3 process see {
4     echo true
5     publishDir params.publish, overwrite:true, mode:'move'
6
7     input:
8     file bim from data
9
10    output:
11    file count
12
13    """
14    hostname
15    echo "Path is \$( pwd )\n "
16    echo "Parent directory has \$( ls .. )\n"
17    echo "My home directory has \$( ls /home/scott )\n"
18    wc -l $bim > count
19    ls
20    """
21 }
```

```
N E X T F L O W ~ version 0.21.2
Launching show_env.nf
[warm up] executor > local
[94/597f09] Submitted process > see (1)
89ad448ae0b2
Path is /home/scott/witsGWAS/dockerized/work/94/597f09ca6cc01c7be
Parent directory has 597f09ca6cc01c7be
My home directory has witsGWAS

YRI.bim
count
```

Directory & File access

Note that although the script's `pwd` shows:

```
/home/scott/witsGWAS/dockerized/work/94/597f09ca6cc01c7be
```

- Only these specific directories are mounted
- Only the files in the innermost directory are available

Any absolute paths (other than those used in staging) will result in error.

Profiles

In nextflow.config

```
1 profiles {  
2   ...  
3   docker {  
4     process.container = 'quay.io/banshee1221/h3agwas-plink:latest'  
5     docker.enabled = true  
6   }  
7 }
```

Now can run as:

```
nextflow run gwas.nf -profile docker
```

This can be extended in many ways

- Different processes can use different containers
- Can mount other host directories
- Can pass arbitrary Docker parameters

Executors

Executors

Executors

Executors

A Nextflow *executor* is the mechanism which Nextflow runs the code in each of the processes

- Default is `local`: process is run as a script

Many others

- PBS/Torque
- SLURM
- Amazon (AWS Batch)
- SGE (Sun Grid Engine)

Selecting an executor Annotating each process

- `executor` directive, e.g. `executor 'pbs'`
- resource constraints

Or, `nextflow.config` file

- either global or per-process

Executors

Nextflow on a cluster (HPC)

Running Nextflow on a cluster (HPC)

Script runs on the *head* node

- Nextflow uses the **executor** information to decide how the job should run
- Each process can be handled differently
- Nextflow submits each process to the job scheduler on your behalf (e.g, if using PBS/Torque, **qsub** is done)

Example

```
1 process {  
2   executor = 'pbs'  
3   queue = 'batch'  
4   scratch = true  
5   cpus = 5  
6   memory = '2GB'  
7 }
```

Executors

Scheduler + Docker

Scheduler + Docker

```
1 process.container = 'quay.io/banshee1221/h3agwas-plink:latest'
2 docker.enabled = false
3
4 process {
5     executor = 'pbs'
6     queue = 'batch'
7     scratch = true
8     cpus = 5
9     memory = '2GB'
10 }
```

Executors

Amazon EC2

Amazon EC2

Netflow has native support for EC2

- You need an account on EC2
- Image (AMI) with the appropriate support

Launch your code:

```
nextflow cloud create GenomeCloud -c 5
```

If successful, Nextflow will give you the name of the headnode of your cluster

- **ssh** into into it
- run Nextflow on it.

Afterwards shut down:

```
nextflow shutdown GenomeCloud
```

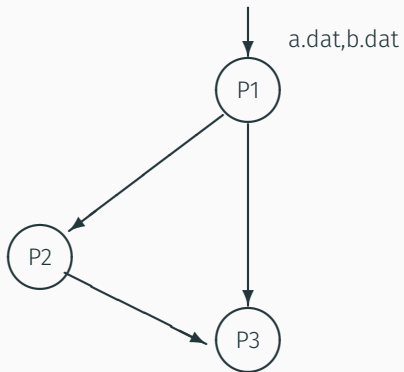
Channel Operations

Channel operations

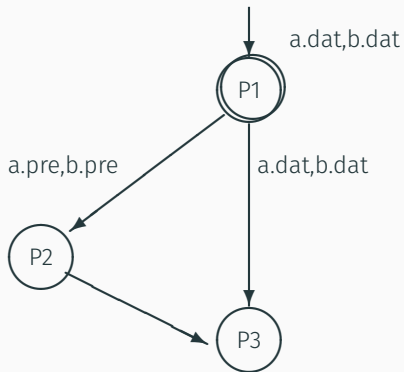
Nextflow tries to maximise concurrency

- processes are by default synchronised by channels
- when data arrives on all input channels, process executes

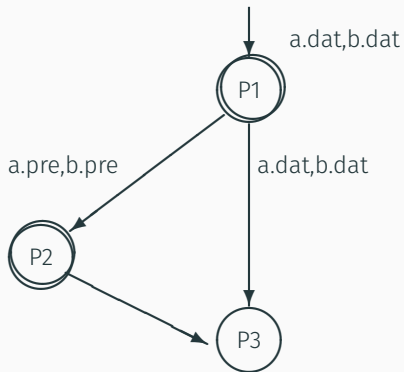
Channel operations



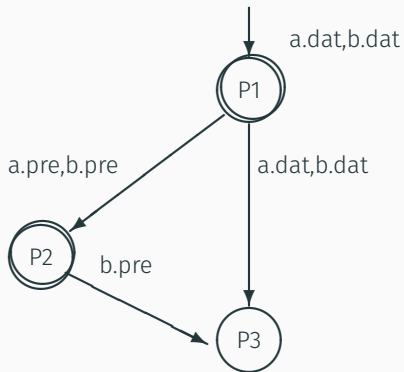
Channel operations



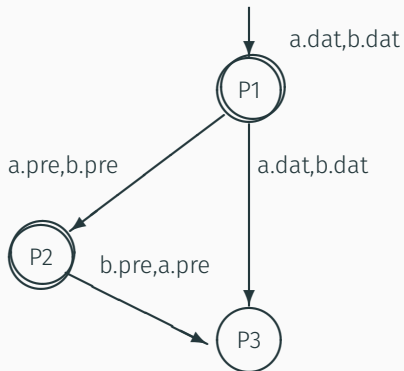
Channel operations



Channel operations



Channel operations



Channel operations

```
1 Channel.fromPath("data/*.dat").set { data }
2
3 process P1 {
4   input:
5     file(data)
6
7   output:
8     file "${fbase}.pre" into channelA
9     file data      into channelB
10
11   script:
12     fbase=data.baseName
13     "echo dummy > ${fbase}.pre"
14 }
15
16 process P2 {
17   input:
18     file pre from channelA
19
20   output:
21     file pre into channelC
22
23   script:
24     if (pre.baseName == "a")
25       "sleep 4"
26     else
27       "sleep 1"
28 }
```

Channel operations

```
1 Channel.fromPath("data/*.dat").set { data }
2
3 process P1 {
4     input:
5     file(data)
6
7     output:
8     file "${fbase}.pre" into channelA
9     file data      into channelB
10
11    script:
12    fbase=data.baseName
13    "echo dummy > ${fbase}.pre"
14 }
15
16 process P2 {
17     input:
18     file pre from channelA
19
20     output:
21     file pre into channelC
22
23    script:
24    if (pre.baseName == "a")
25        "sleep 4"
26    else
27        "sleep 1"
28 }
```

Try

```
29 process P3 {
30     echo true
31
32     input:
33     file(data) from channelB
34     file(pre)  from channelC
35
36     script:
37     """
38     echo "${data} - $pre"
39     """
40 }
```

Channel operations

Solution: `join/merge` channels

- `x.merge(y)`

Items emitted by the channels `x` and `y` are combined into a new channel.

- `x.join(y)`

Items emitted by the channels `x` and `y` are joined together into one channel based on existing matching key. Default: first element in each item.

Channel Operations

Using `join`

Using join

```
1 ch1 = Channel.from( "a","b","c" )
2 ch2 = Channel.from( "a","d","e","a","c","b" )
3 ch1.join(ch2).subscribe { println it }
```

```
a
b
c
```

Using join

```
1 ch1 = Channel.from( "a","b","c" )
2 ch2 = Channel.from( "a","d","e","a","c","b" )
3 ch1.join(ch2).subscribe { println it }
```

```
a
b
c
```

Tuples:

```
1 ch1 = Channel.from( ["a",1], ["b",4], ["c",5] )
2 ch2 = Channel.from( ["a",10], ["d",8], ["e",7], ["a",9], ["c",1], ["b",10] )
3 ch1.merge(ch2).subscribe { println it }
```

```
[a, 1, 10]
[b, 4, 10]
[c, 5, 1]
```

Channel Operations

Using merge

Using merge

```
1 ch1 = Channel.from( "a","b","c" )  
2 ch2 = Channel.from( "a","d","e","a","c","b" )  
3 ch1.merge(ch2).subscribe { println it }
```

```
[a, a]  
[b, d]  
[c, e]
```

Using merge

```
1 ch1 = Channel.from( "a","b","c" )  
2 ch2 = Channel.from( "a","d","e","a","c","b" )  
3 ch1.merge(ch2).subscribe { println it }
```

```
[a, a]  
[b, d]  
[c, e]
```

Tuples:

```
1 ch1 = Channel.from( ["a",1], ["b",4], ["c",5] )  
2 ch2 = Channel.from( ["a",10], ["d",8], ["e",7], ["a",9], ["c",1], ["b",10] )  
3 ch1.merge(ch2).subscribe { println it }
```

```
[a, 1, a, 10]  
[b, 4, d, 8]  
[c, 5, e, 7]
```

Channel Operations

`join` vs `merge`

join vs merge

join

- If values are singletons, then the values must be the same
- If value is tuple if the, then the first element of the tuple must be the same

merge

- Merges everything into a channel, no matching.

Channel Operations

Working version of the example

Working version of the example

```
1 Channel.fromPath("data/*.dat").set { data }
2
3 process P1 {
4     echo true
5
6     input:
7     file(data)
8
9     output:
10    set val(data.baseName), file("${fbase}.pre") into channelA
11    set val(data.baseName), file(data) into channelB
12
13    script:
14    fbase=data.baseName
15    "echo dummy > ${fbase}.pre"
16
17 process P2 {
18     echo true
19
20     input:
21     set name, file(pre) from channelA
22
23     output:
24     set name, file(pre) into channelC
```

```
25
26     script:
27     if (pre.baseName = /*.TMP.*/)
28         "sleep 4"
29     else
30         "sleep 1"
31     }
32 process P3 {
33     echo true
34
35     input:
36     set name, file(data), file(pre) from channelB.join(channelC)
37
38     script:
39     """
40     echo "${data} - ${pre}"
41     """
42 }
```

Channel Operations

Copying channels

Copying channels

You often need to copy a channel

```
1  process do {  
2      ..  
3  
4      output:  
5      file ("x.*") into out_ch  
6  
7      ..  
8  }  
9  
10 out_ch.separate(a_ch, b_ch, c_ch)
```

Copying channels

You often need to copy a channel

```
1 process do {  
2     ..  
3  
4     output:  
5     file ("x.*") into out_ch  
6  
7     ..  
8 }  
9  
10 out_ch.separate(a_ch, b_ch, c_ch)
```

Alternatively

```
1 process do {  
2     ..  
3  
4     output:  
5     file ("x.*") into (a_ch, b_ch, c_ch)  
6  
7     ..  
8 }
```