Nextflow:
a tutorial through examples

Phelelani Mpangase

22 August 2019

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

CODATA

Outline

Introduction to Nextflow

Introduction to Nextflow

Introduction

Resources

- https://github.com/fpsom/CODATA-RDA-Advanced-Bioinformatics-2019/blob/phele/4.Day4.md

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

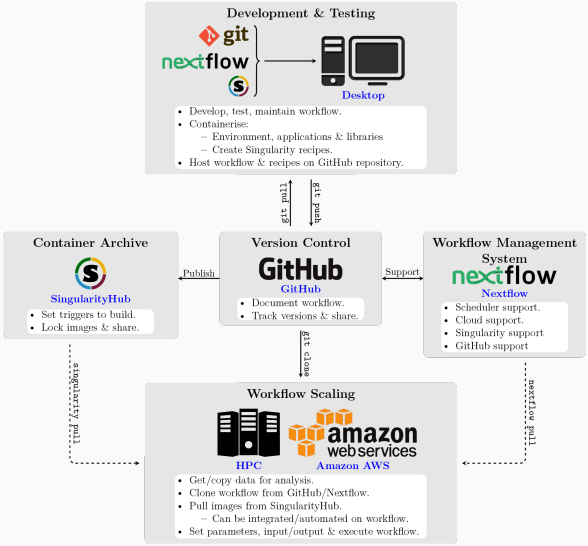
Workflow Languages

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 nexflow
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`

```
#!/usr/bin/env nexflow

input_ch = Channel.fromPath("data/11.bim")

process getIDs {
  input:
  file input from input_ch

  output:
  file "ids" into id_ch
  file "11.bim" into orig_ch

  script:
  "cut -f 2 $input | sort > ids"
}

process getDups {
    input:
    file input from id_ch

    output:
    file "dups" into dups_ch

    script:
    """
    uniq -d $input > dups
    touch ignore
    """
}
```

```
process removeDups {
    input:
    file badids from dups_ch
    file orig from orig_ch

    output:
    file "clean.bim" into output

    script:
    "grep -v -f $badids $orig > clean.bim "
}

output.subscribe { print "Done!" }
```

Simple Example: Using `nextflow`

```
#!/usr/bin/env nexflow

input_ch = Channel.fromPath("data/11.bim")

process getIDs {
  input:
  file input from input_ch

  output:
  file "ids" into id_ch
  file "11.bim" into orig_ch

  script:
  "cut -f 2 $input | sort > ids"
}

process getDups {
    input:
    file input from id_ch

    output:
    file "dups" into dups_ch

    script:
    """
    uniq -d $input > dups
    touch ignore
    """
}
```

```
process removeDups {
    input:
    file badids from dups_ch
    file orig from orig_ch

    output:
    file "clean.bim" into output

    script:
    "grep -v -f $badids $orig > clean.bim "
}

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

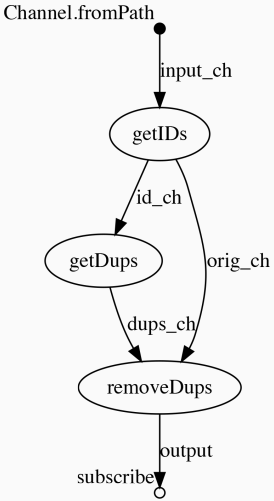
Visualising the Workflow

Visualising the Workflow

Nextflow supports several visualisation tools:

-with-dag

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



Visualising the Workflow

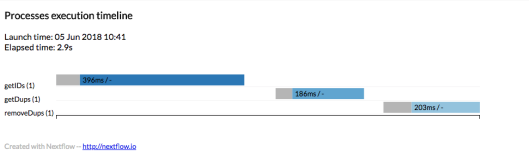
Nextflow supports several visualisation tools:

-with-dag

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

-with-timeline

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



Visualising the Workflow

Nextflow supports several visualisation tools:

-with-dag

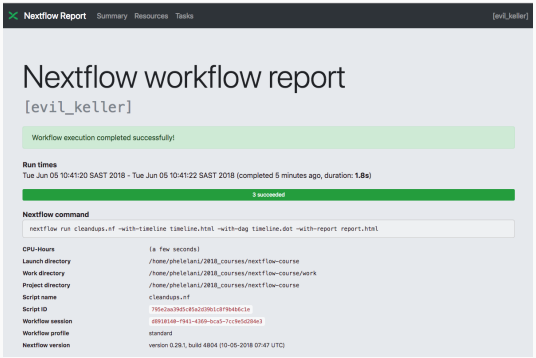
```
nextflow run cleandups.nf -with-dag <file-name>
```

-with-timeline

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

-with-report

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



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

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

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

Generalising and Extending

Generalising Our Example

Workflow: Multiple Inputs

```
params.data_dir = "data"
input_ch = Channel.fromPath("${params.data_dir}/*.bim")

process getIDs {
    input:
    file input from input_ch

    output:
    file "${input.baseName}.ids" into id_ch
    file "$input" into orig_ch

    script:
    "cut -f 2 $input | sort > ${input.baseName}.ids"
}

process getDups {
    input:
    file input from id_ch

    output:
    file "${input.baseName}.dups" into dups_ch

    script:
    out = "${input.baseName}.dups"
    """
    uniq -d $input > $out
    touch ignore
    """
}
```

Workflow: Multiple Inputs

```
params.data_dir = "data"
input_ch = Channel.fromPath("${params.data_dir}/*.bim")

process getIDs {
    input:
    file input from input_ch

    output:
    file "${input.baseName}.ids" into id_ch
    file "$input" into orig_ch

    script:
    "cut -f 2 $input | sort > ${input.baseName}.ids"
}

process getDups {
    input:
    file input from id_ch

    output:
    file "${input.baseName}.dups" into dups_ch

    script:
    out = "${input.baseName}.dups"
    """
    uniq -d $input > $out
    touch ignore
    """
}
```

```
process removeDups {
    publishDir "output", pattern: "${badids.baseName}_clean.bim"
      ↪ , overwrite:true, mode:'copy'

    input:
    file badids from dups_ch
    file orig from orig_ch

    output:
    file "${badids.baseName}_clean.bim" into cleaned_ch

    script:
    "grep -v -f $badids $orig > ${badids.baseName}_clean.bim "
}
```

Workflow: Multiple Inputs

```
params.data_dir = "data"
input_ch = Channel.fromPath("${params.data_dir}/*.bim")

process getIDs {
    input:
    file input from input_ch

    output:
    file "${input.baseName}.ids" into id_ch
    file "$input" into orig_ch

    script:
    "cut -f 2 $input | sort > ${input.baseName}.ids"
}

process getDups {
    input:
    file input from id_ch

    output:
    file "${input.baseName}.dups" into dups_ch

    script:
    out = "${input.baseName}.dups"
    """
    uniq -d $input > $out
    touch ignore
    """
}
```

```
process removeDups {
    publishDir "output", pattern: "${badids.baseName}_clean.bim"
        ↪ , overwrite:true, mode:'copy'

    input:
    file badids from dups_ch
    file orig from orig_ch

    output:
    file "${badids.baseName}_clean.bim" into cleaned_ch

    script:
    "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:

```
splits = [400,500,600]

process splitIDs  {
    input:
    file bim from cleaned_ch
    each split from splits

    output:
    file ("*-$split-*") into output_ch;

    script:
    "split -l $split $bim ${bim.baseName}-$split- "
}
```

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

```
#!/usr/bin/env nextflow
params.dir = "data/pops/"
dir = params.dir
params.pops = ["YRI","CEU","BEB"]

Channel
    .from(params.pops)
    .map { pop ->
        [ file("$dir/${pop}.bed"),
          file("$dir/${pop}.bim"),
          file("$dir/${pop}.fam")]
    }
    .set { plink_data }

plink_data.subscribe { println "$it" }
```

Grouped Files - Version 1: map

```
#!/usr/bin/env nextflow
params.dir = "data/pops/"
dir = params.dir
params.pops = ["YRI","CEU","BEB"]

Channel
    .from(params.pops)
    .map { pop ->
        [ file("$dir/${pop}.bed"),
          file("$dir/${pop}.bim"),
          file("$dir/${pop}.fam")]
    }
    .set { plink_data }

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

```
#!/usr/bin/env nextflow
params.dir = "data/pops/"
dir = params.dir
params.pops = ["YRI","CEU","BEB"]

Channel
    .from(params.pops)
    .map { pop ->
        [ file("$dir/${pop}.bed"),
          file("$dir/${pop}.bim"),
          file("$dir/${pop}.fam")]
    }
    .set { plink_data }

plink_data.subscribe { println "$it" }
```

```
process getFreq {
  input:
    set file(bed), file(bim), file(fam) from plink_data
  output:
    file "${bed.baseName}.frq" into result

  """
  plink --bed $bed \
    --bim $bim \
    --fam $fam \
    --freq \
    --out ${bed.baseName}"
  """
}
```

```
[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)

```
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" }
```

```
[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]]
```

Grouped Files - Version 2: `fromFilePairs`

```
process checkData {
    input:
    set pop, file(pl_files) from plink_data

    output:
    file "${pl_files[0]}.frq" into result

    script:
    base = pl_files[0].baseName
    "plink --bfile $base --freq --out ${base}"
}
```

Grouped Files - Version 2: `fromFilePairs`

```
process checkData {
    input:
    set pop, file(pl_files) from plink_data

    output:
    file "${pl_files[0]}.frq" into result

    script:
    base = pl_files[0].baseName
    "plink --bfile $base --freq --out ${base}"
}
```

```
process checkData {
    input:
    set pop, file(pl_files) from plink_data

    output:
    file "${pop}.frq" into result

    script:
    "plink --bfile $pop --freq  --out $pop"
}
```

Grouped Files - Version 2: `fromFilePairs`

```
process checkData {
    input:
    set pop, file(pl_files) from plink_data

    output:
    file "${pl_files[0]}.frq" into result

    script:
    base = pl_files[0].baseName
    "plink --bfile $base --freq --out ${base}"
}
```

```
process checkData {
    input:
    set pop, file(pl_files) from plink_data

    output:
    file "${pop}.frq" into result

    script:
    "plink --bfile $pop --freq  --out $pop"
}
```

Grouped Files - Final Version

```groovy
Channel.fromPath("data/*.dat").set { data }

process P1 {
    echo true
    input:
    file(data)

    output:
    set val(data.baseName), file("${fbase}.pre") into channelA
    set val(data.baseName), file(data) into channelB

    script:
    fbase=data.baseName
    "echo dummy > ${fbase}.pre"
}

process P2 {
    echo true
    input:
    set name, file(pre) from channelA

    output:
    set name, file(pre) into channelC

    script:
    if (pre.baseName = /.*TMP.*/)
        "sleep 4"
    else
        "sleep 1"
}
```

```groovy
process P3 {
    echo true

    input:
    set name, file(data), file(pre) from channelB.join(channelC)

    """
    echo "${data} - ${pre}"
    """
}
```

Generalising and Extending

On absolute paths

Absolute paths

```
input = Channel.fromPath("/data/batch1/myfile.fa")

process show {
    input:
    file data from input

    output:
    file 'see.out'

    script:
    cp $data /home/scott/answer
    ...
```

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("/popdata/1k-2014/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

```
profiles {
  ...
  docker {
    process.container = 'quay.io/banshee1221/h3agwas-plink:latest'
    docker.enabled = true
  }
}
```

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 Nexflow 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

```
process {
  executor = 'pbs'
  queue = 'batch'
  scratch = true
  cpus = 5
  memory = '2GB'
}
```

Executors

Scheduler + Docker

Scheduler + Docker

```
process.container = 'quay.io/banshee1221/h3agwas-plink:latest'
docker.enabled = false

process {
  executor = 'pbs'
  queue = 'batch'
  scratch = true
  cpus = 5
  memory = '2GB'
}
```

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
```

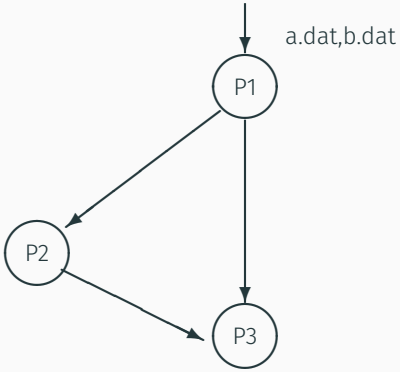
Channel Operations

Channel operations

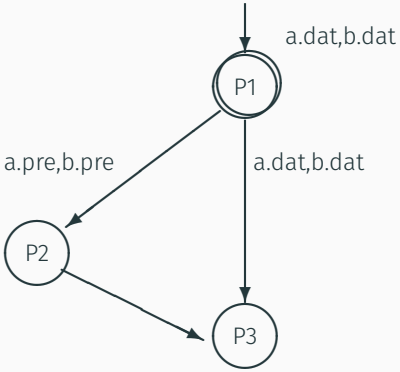
Nextflow tries to maximise concurrency

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

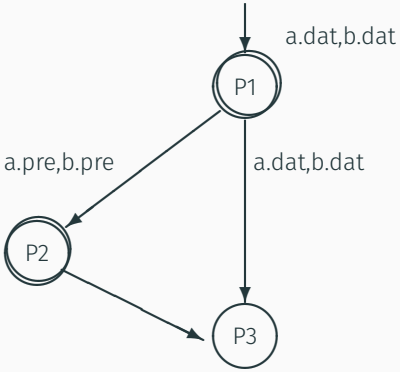
Channel operations



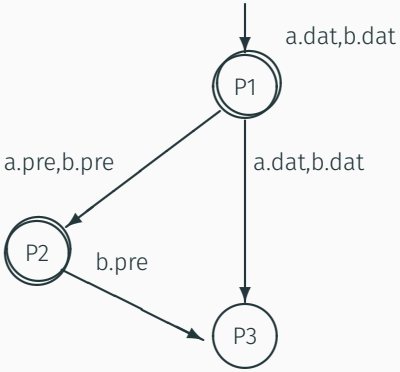
Channel operations



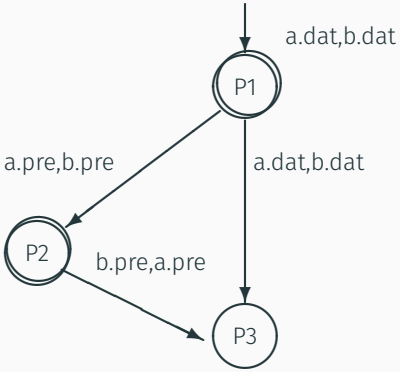
Channel operations



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

```
Channel.fromPath("data/*.dat").set { data }

process P1 {
    input:
    file(data)

    output:
    file  "${fbase}.pre" into channelA
    file  data           into channelB

    script:
    fbase=data.baseName
    "echo dummy > ${fbase}.pre"
}

process P2 {
    input:
    file pre from channelA

    output:
    file pre into channelC

    script:
    if (pre.baseName == "a")
      "sleep 4"
    else
      "sleep 1"
}
```

Try

```
process P3 {
    echo true

    input:
    file(data) from channelB
    file(pre)  from channelC

    script:
    """
    echo "${data} - $pre"
    """
}
```

Channel operations

Solution: `join`/`merge` channels

- `x.merge(y)`
  Items emmited by the channels *x* and *y* are combined into a new channel.
- `x.join(y)`
  Items emmited 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
```

```
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]
```

```
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     script:
26     if (pre.baseName = /.*TMP.*/)
27       "sleep 4"
28     else
29       "sleep 1"
30 }
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

```
process do {
    ..

    output:
    file ("x.*") into out_ch

    ..
}

out_ch.separate(a_ch, b_ch, c_ch)
```

Copying channels

You often need to copy a channel

```
process do {
    ..

    output:
    file ("x.*") into out_ch

    ..
}

out_ch.separate(a_ch, b_ch, c_ch)
```

Alternatively

```
process do {
    ..

    output:
    file ("x.*") into (a_ch, b_ch, c_ch)

    ..
}
```

H3AVarCall

H3AVarCall: Hands-on Variant Calling Practical

Prepare your workspace for the variant calling workflow!

```
## Change directory to your day4 working folder:
cd ~/Documents/day4

## Clone the H3AVarCall repository from GitHub:
git clone https://github.com/h3abionet/h3avarcall.git

## Change directory to the repository:
cd h3avarcall

## Create symbolic links to the Singularity images with applications:
ln -s /home/nfs3/h3avarcall/containers/* containers/

## Make a temorary folder in the 'scratch directory' for your 'work' folder:
mkdir -p /scratch/<USERNAME>/work
```

Lets look at some important files:

- `main.nf`
- `main.config`
- `nextflow.config`

DONE!! Now we are ready to start with the variant calling analysis!

H3AVarCall

Quality Checks - FastQC

Quality Checks - FastQC

Time allocated for this step: **10 minutes** Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.QC
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--1_QC
  |  |  |--workflow_report
  |  |  |  |--h3avarcall_report.html
  |  |  |  |--h3avarcall_timeline.html
  |  |  |  |--h3avarcall_workflow.dot
  |  |  |  |--h3avarcall_trace.txt
  |  |  |--<sample_1>_R1.fastqc.html .. <sample_N>_R1.fastqc.html
  |  |  |--<sample_1>_R2.fastqc.html .. <sample_N>_R1.fastqc.html
```

H3AVarCall

Read Trimming - Trimmomatic

Read Trimming - Trimmomatic

Time allocated for this step: **10 minutes**

Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.ReadTrimming
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--2_Read_Trimming
  |  |  |--workflow_report
  |  |  |  |--h3avarcall_report.html
  |  |  |  |--h3avarcall_timeline.html
  |  |  |  |--h3avarcall_workflow.dot
  |  |  |  |--h3avarcall_trace.txt
  |  |  |--<sample_1>.1P.fastq.gz .. <sample_N>.1P.fastq.gz
  |  |  |--<sample_1>.2P.fastq.gz .. <sample_N>.2P.fastq.gz
```

H3AVarCall

Read Alignment - **BWA**, **GATK** and **Samtools**

Read Alignment - BWA, GATK and Samtools

Time allocated for this step: **10 minutes**

Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.ReadAlignment
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--3_Read_Alignment
  |  |  |--workflow_report
  |  |  |  |--h3avarcall_report.html
  |  |  |  |--h3avarcall_timeline.html
  |  |  |  |--h3avarcall_workflow.dot
  |  |  |  |--h3avarcall_trace.txt
  |  |  |--<sample_1>_md.recal.bam .. <sample_N>_md.recal.bam
  |  |  |--<sample_1>_md.recal.bai .. <sample_N>_md.recal.bai
```

H3AVarCall

Variant Calling - **GATK**

Variant Calling - GATK

Time allocated for this step: **60 minutes**

Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.VariantCalling
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--4_Variant_Calling
  |  |  |--workflow_report
  |  |  |  |--h3avarcall_report.html
  |  |  |  |--h3avarcall_timeline.html
  |  |  |  |--h3avarcall_workflow.dot
  |  |  |  |--h3avarcall_trace.txt
  |  |  |--chr_1_genotyped.vcf.gz .. chr_22_genotyped.vcf.gz
  |  |  |--chr_1_genotyped.vcf.gz.tbi .. chr_22_genotyped.vcf.gz.tbi
```

H3AVarCall

Variant Filtering - GATK

Variant Filtering - GATK

Time allocated for this step: **10 minutes**

Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.VariantFiltering
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--5_Variant_Filtering
  |  |  |--workflow_report
  |  |  |  |--h3avarcall_report.html
  |  |  |  |--h3avarcall_timeline.html
  |  |  |  |--h3avarcall_workflow.dot
  |  |  |  |--h3avarcall_trace.txt
  |  |  |--genome.SNP-recal.vcf.gz
  |  |  |--genome.SNP-recal.vcf.gz.tbi
```

H3AVarCall

Quality Checks - MultiQC

Quality Checks - MultiQC

Time allocated for this step: **10 minutes**

Run:

```
nextflow run main.nf -profile local -w /scratch/<USERNAME>/work --mode do.MultiQC
```

Results:

```
h3avarcall
  |--variant_calling_results
  |  |--MultiQC
  |  |  |--multiqc_data
  |  |  |--multiqc_report.html
```