GET STARTED WITH NEXTFLOW DSL2

Paolo Di Tommaso, Seqera Labs

seqeralabs

nextflow

DSL2 IN A NUTSHELL

- Evolutionary extension of Nextflow syntax
- Enable the writing of reusable tasks and sub-workflows
- Allows more concise and expressive definition of pipeline logic
- Easy migration of existing Nextflow processes
- Do not change the dataflow programming paradigm

TASKS COMPOSITION

```
process foo {
  output:
    path 'foo.txt' into foo_ch
  script:
    """
    your_command > foo.txt
    """
}

process bar {
  input:
    path x from foo_ch
  output:
    path 'bar.txt' into bar_ch
  """
    another_command $x > bar.txt
  """
}
```

TASKS COMPOSITION

```
process foo {
  output:
    path 'foo.txt' into foo_ch
  script:
    """
    your_command > foo.txt
    """
}

process bar {
  input:
    path x from foo_ch
  output:
    path 'bar.txt' into bar_ch
  """
    another_command $x > bar.txt
  """
}
```

MODULAR TASKS

```
process foo {
  output:
    path 'foo.txt'
  script:
    """
    your_command > foo.txt
    """
}

process bar {
  input:
    path x
  output:
    path 'bar.txt'
  """
    another_command $x > bar.txt
  """
}
```

WELCOME DSL-2 !

```
nextflow.enable.dsl=2

process foo {
  output:
    path 'foo.txt'
  script:
    """
    your_command > foo.txt
    """
}

process bar {
  input:
    path x
  output:
    path 'bar.txt'
  """
    another_command $x > bar.txt
  """
}


workflow {
   foo()
   bar( channel.of('file1.txt','file2.txt') )
}
```

PROCESS OUTPUT

```
process foo {
    output:
      path 'foo.txt'
    script:
      """
      your_command > foo.txt
      """
}

process bar {
    input:
      path x
    output:
      path 'bar.txt'
    script:
      """
      another_command $x > bar.txt
      """
}

workflow {
    foo()
    bar( foo.out )
    bar.out.view()
}
```

NAMED OUTPUT

```
process foo {
  output:
    path '*.bam', emit: samples_bam

  '''
  your_command --here
  '''
}


workflow {
    foo()
    foo.out.samples_bam.view()
}
```

HANDS-ON

MODULES !

```
process foo {
    input:
      path x
    output:
      path 'foo.txt'
    script:
      """
      your_command $x > foo.txt
      """
}
```

```
include { foo } from './some/module.nf'


workflow {
  data = channel.fromPath('/some/data/*.txt')
  foo(data)
}
```

MULTIPLE INCLUSIONS

```
include { foo; bar } from './some/module'



workflow {
    data = channel.fromPath('/some/data/*.txt')
    foo(data)
    bar(data)
}
```

MODULE ALIASES

```
include { foo } from './modules/this'
include { foo as bar } from './modules/that'



workflow {
    foo(some_data)
    bar(other_data)
}
```

MODULE PARAMETERS

```
params.foo = 'Hello'
params.bar = 'world!'


process sayHello {
  script:
  """
  $params.foo $params.bar
  """
}
```

```
params.foo = 'Hola'
params.bar = 'Mundo'

include{ sayHello } from './some/module'


workflow {
    sayHello()
}
```

OVERRIDE PARAMETERS

```
params.foo = 'Hello'
params.bar = 'world!'


process sayHello {
  script:
  """
  $params.foo $params.bar
  """
}
```

```
include{ sayHello } from './some/module' addParams(foo: 'Ciao')


workflow {
    sayHello()
}
```

HANDS-ON

SUB-WORKFLOWS

```
workflow my_pipeline {
  foo()
  bar( foo.out.collect() )
}
```

```
include { my_pipeline } from './my-module.nf'

workflow {
  my_pipeline()
}
```

WORKFLOW PARAMS & INPUTS

```
params.data = '/some/data/file'


workflow flow_1 {
    if( params.data )
        bar(params.data)
    else
        bar(foo())
}




workflow flow_2 {
    take: data
    main:
    foo(data)
    bar(foo.out)
}
```

WORKFLOW OUTPUTS

```
workflow flow_1 {
  main:
    foo()
    bar(foo.out)
  emit:
    bar.out
}

workflow flow_2 {
  main:
    foo()
    bar(foo.out)
  emit:
    my_data = bar.out
}

workflow {
  flow_1()
  flow_1.out.view()

  flow_2()
  flow_2.out.my_data.view()
}
```

HANDS-ON

THE PIPE OPERATOR !

```
process foo {
    input: val data
    output: val result
    exec:
    result = "$data world"
}


workflow {
   channel.from('Hello','Hola','Ciao') | foo | map { it.toUpperCase() } | view
}
```

THE PIPE OPERATOR !

```
process foo {
    input: val data
    output: val result
    exec:
    result = "$data world"
}


workflow {
   channel.from('Hello','Hola','Ciao') \
    | foo \
    | map { it.toUpperCase() } \
    | view
}
```

STUB-RUN FEATURE

WHAT'S THE PROBLEM WITH DRY-RUN?

```
process foo {

   input:
     path some_data
   output:
     path '*.txt'
   script:
     """
     any_command_here -in $some_data
     """
}


workflow {
   data = channel.fromPath('data/*.fastq')
   foo(dta)
}
```

INTRODUCING STUB-RUN!

```
process foo {
   input:
     path some_data
   output:
     path '*.txt'
   script:
     """
     any_command_here -in $some_data
     """
   stub:
     """
     echo "fake data" > alpha.txt
     echo "more data" > omega.txt
     """
}

workflow {
   data = channel.fromPath('data/*.fastq')
   foo(dta)
}
```

```
» nextflow run your-script.nf -stub
```

Q&A