Good Practices

How to avoid race condition

```
Channel.of(1,2,3) | map { it -> X=it; X+=2 } | view { "ch1 = $it" }
Channel.of(1,2,3) | map { it -> X=it; X*=2 } | view { "ch2 = $it" }
```

Using the def keyword makes the variable local to the enclosing scope

omitting the def keyword makes the variable global to the entire script.

```
// local variable
Channel.of(1,2,3) | map { it -> def X=it; X+=2 } | view { "ch1 = $it" }
// global variable
Channel.of(1,2,3) | map { it -> X=it; X*=2 } | view { "ch2 = $it" }
```

example10.nf

Good Practices - Metadata

Use runtime and nextflow metadata

```
workflow.commandLine
workflow.commitId
workflow.complete
workflow.homeDir
workflow.launchDir
...
```

```
nextflow.build
nextflow.version
nextflow.timestamp
```

```
workflow.onComplete {
    println "Pipeline completed at: $workflow.complete"
    println "Execution status: ${ workflow.success ? 'OK' : 'failed' }"
}
```

```
workflow.onError {
    println "Error: Pipeline execution error: ${workflow.errorMessage}"
}
```

Good Practices

Resuming a pipeline is a good test

Resuming help you detect reproducibility and channel error

Remove publishDir when pipeline fails with workflow.OnError

Good Practices - Templates

https://nf-co.re/

https://github.com/bioinfo-pf-curie/geniac

Geniac: Automatic Configuration GENerator and Installer for nextflow pipelines.

Good Practices - Templates

```
## DSL2

assets/
bin/
conf/
docs/
recipes/
test/
nf-modules/common/process
nf-modules/common/subworkflow
nf-modules/local/process
nf-modules/local/subworkflow
main.nf
nextflow.config
```

nextflow.config - the default config file which will load all the others
base.config - default config – no need to change it
cluster.config - define which cluster/scheduler to use
genome.config - define all annotation files (from /data/annotation/pipelines)
process.config - define the RAM/CPU for each labels. Can be customize for some specific process

Good Practices - Portability

Simplify: 1 process =1 task
Move when, publishDir, and options in config file

```
process bwaMem{
  tag "${meta.id}"
  label 'bwa'
  label 'highCpu'

  input:
  tuple val(meta), path(reads)
  path(index)

  output:
  tuple val(meta), path("*.sam"), emit: sam
  path("versions.txt"), emit: versions

  when:
  task.ext.when == null || task.ext.when

  script:
  def args = task.ext.args ?: ''
  def prefix = task.ext.prefix ?: "${meta.id}"

  """
  localIndex=`find -L ./ -name "*.amb" | sed 's/.amb//'`
  refName=`basename \${localIndex}`

  bwa mem $params.bwaOpts $args -t $task.cpus \${localIndex} $reads

  echo "Bwa-mem "\$(bwa 2>&1 | grep Version | cut -d" " -f2) &> versions.txt

  """
}
```

Module.config

```
withName:'bwaMem' {
    publishDir = [
      [
        path: { "${params.outDir}/mapping/sam/${meta.id}/" },
        mode: 'copy',
        pattern: "*.sam",
        enabled: params.saveAlignedIntermediates
      ]
    ]
    ext.args = {
        params.bwaOpts ?: '',"-R \"@RG\\tID:${meta.id}"
    }
    ext.prefix = { "${meta.part} > 1" ? "${meta.id}_part${meta.chunk}" : "${meta.id}" }
  }
```

Process.config

```
withLabel: minCpu {
    cpus = { checkMax( 1 * task.attempt, 'cpus' ) }
  }
  withLabel: lowCpu {
    cpus = { checkMax( 2 * task.attempt, 'cpus' ) }
  }
  withLabel: medCpu {
    cpus = { checkMax( 4 * task.attempt, 'cpus' ) }
  }
  withLabel: highCpu {
    cpus = { checkMax( 8 * task.attempt, 'cpus' ) }
  }
```

INSTITUT FRANÇAIS DE BIOINFORMATIQUE elixir FRANCE institut Curie

Good Practices - Portability

```
include bwaMem from '../process/bwa'
include samtoolsSort from '../process/samtools'

workflow mapping {

    take:
    reads
    index

    main:
    bwaMem(
        reads,
        index
    )

    samtoolsSort(
        bwaMem.out.bam
    )

    emit:
    bam = samtoolsSout.out.bam
}
```

```
include bwaMem from '../subworkflow/mapping'
include samtoolsSort from '../process/fastqc'

workflow {

    chReads     = ...
    chBwaIndex  = ...

    main:
    fastqc( chReads )

    mapping(
        chReads,
        chBwaIndex.collect()
    )
}
```

Use and re-use sub-workflow

Container - reproducibility

Nextflow support docker/singularity/apptainer

Rule is one container per process

Nextflow will launch apptainer exec

```
nextflow run <your script> -with-apptainer [apptainer image file]
```

or

nextflow.config

```
process.container = '/path/to/apptainer.img'
apptainer.enabled = true
```

nextflow.config or singularity.config

```
process {
    withLabel:foo {
        container = 'image_name_1'
    }
    withLabel:bar {
        container = 'image_name_2'
    }
}

Apptainer {

    autoMounts = true
    enabled = true
    runOptions="--containall"
}
```

Good Practices - Reproducibility

How to avoid non-deterministic process inputs

```
workflow {
    ch_foo = Channel.of( ['1', '1.foo'], ['2', '2.foo'] )
    ch_bar = Channel.of( ['2', '2.bar'], ['1', '1.bar'] )
    ch_foo.merge(ch_bar)
}
```

merging the inputs and is not deterministic

Join will use the key

```
workflow {
    ch_foo = Channel.of( ['1', '1.foo'], ['2', '2.foo'] )
    ch_bar = Channel.of( ['2', '2.bar'], ['1', '1.bar'] )
    ch_foo.join(ch_bar))
}
```

Good Practices - Plugins

```
plugin { id 'nf-boost'}
boost { cleanup = true}
```

```
plugin { id 'nf-boost'}

if ( params.cleanup == "auto" ) {
    boost.cleanup = true
    boost.cleanupInterval = '180s'
}
```

```
nextflow run <pipeline> --cleanup auto
```

Good Practices - External

You can call groovy class or function from a workflow

```
def goodBy(){
        println 'Process has finished!'
        return null
}
```

```
//import a groovy function
include { goodBy } from './lib/nftools.groovy'
//import a groovy class
println Utils.Hello("main.nf")

workflow {
    println "My great pipeline !!"
    goodBy()
}
```

```
class Utils {
    public static String Hello (String label) {
        return "Hello World from ${label} !"
    }
}
```

example10.nf

Good Practices – recursion (experimental)

How to call a process multiple times

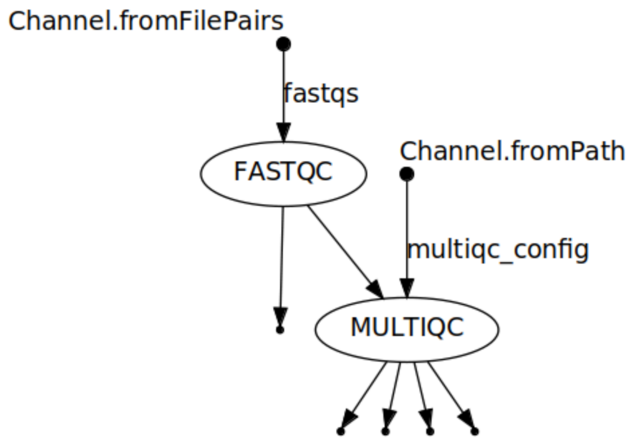
```
nextflow.enable.dsl=2
nextflow.preview.recursion=true

process foo {
  input:
    val x
  output:
    val y
  exec:
    y = x * 2
}

workflow {
    foo.recurse(10).times(4)
    foo.out[0].view()
}
```

example11.nf

TP

Write a pipeline for quality control over paired fastq with multiqc and fastqc.

Use slurm and singularity to run fastqc and multiqc.



INSTITUT FRANÇAIS DE BIOINFORMATIQUE