Simple RNA-Seq workflow

Overview

1. Indexes a transcriptome file
2. Performs quality controls
3. Performs quantification
4. Creates a MultiQC report

Requirements

Softwares: Nextflow-training dir in Gitpod
Languages: Python|Bash
Scripts: script7.nf
Tools: Salmon, FastQC, MultiQC

Links

Video tutorial
https://www.youtube.com/watch?v=nzR8DRq13nE

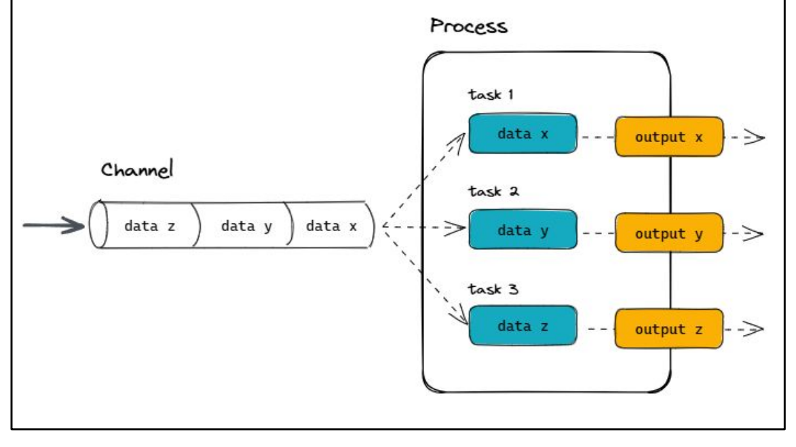
Written tutorial
https://training.nextflow.io/basic_training/rnaseq_pipeline/

Software link
https://nextflowio-training-2hwzxy3c0vu.ws-us117.gitpod.io/

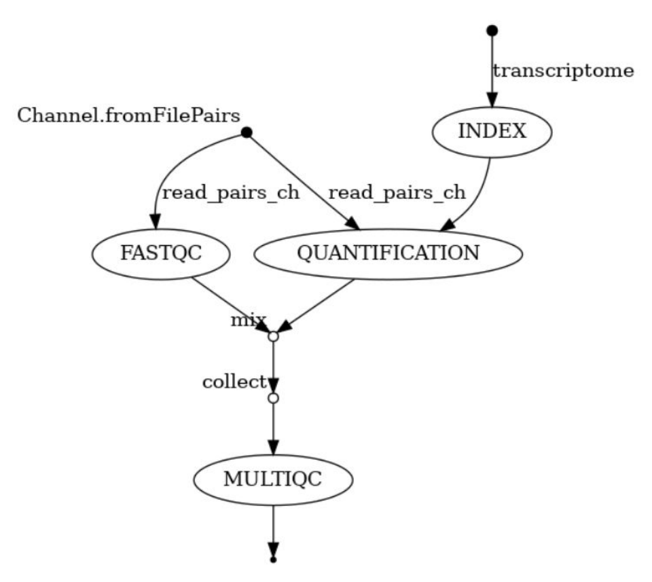
Definitions

A Nextflow workflow is made by joining processes.
Processes can be written in any Linux executable scripting language and are executed independently. Processes only communicate via asynchronous queues called channels.
Every input and output of a processes is a channel. The interaction between these processes are defined by input and output declarations.
Process is defined by input, output and script.

Nextflow pipeline represented as a direct acrylic graph (DAG). Vertices represent pipeline's processes and operators. Edges represent data dependencies (channels) between them.



nf-training -> nextflow run script7.nf -resume --reads 'data/ggal/*_{1,2}.fq' **-with-dag flowchart.png**

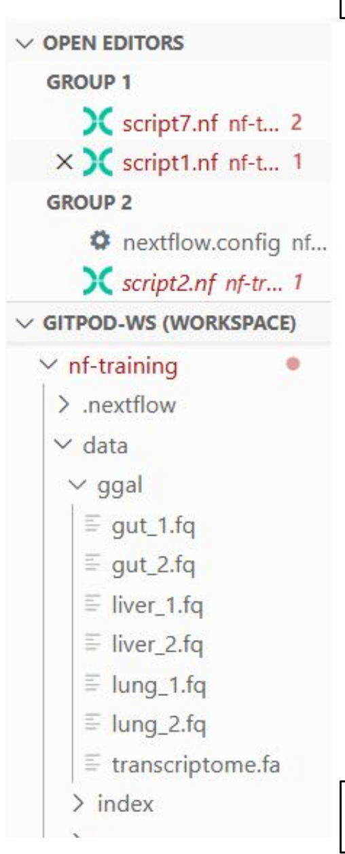


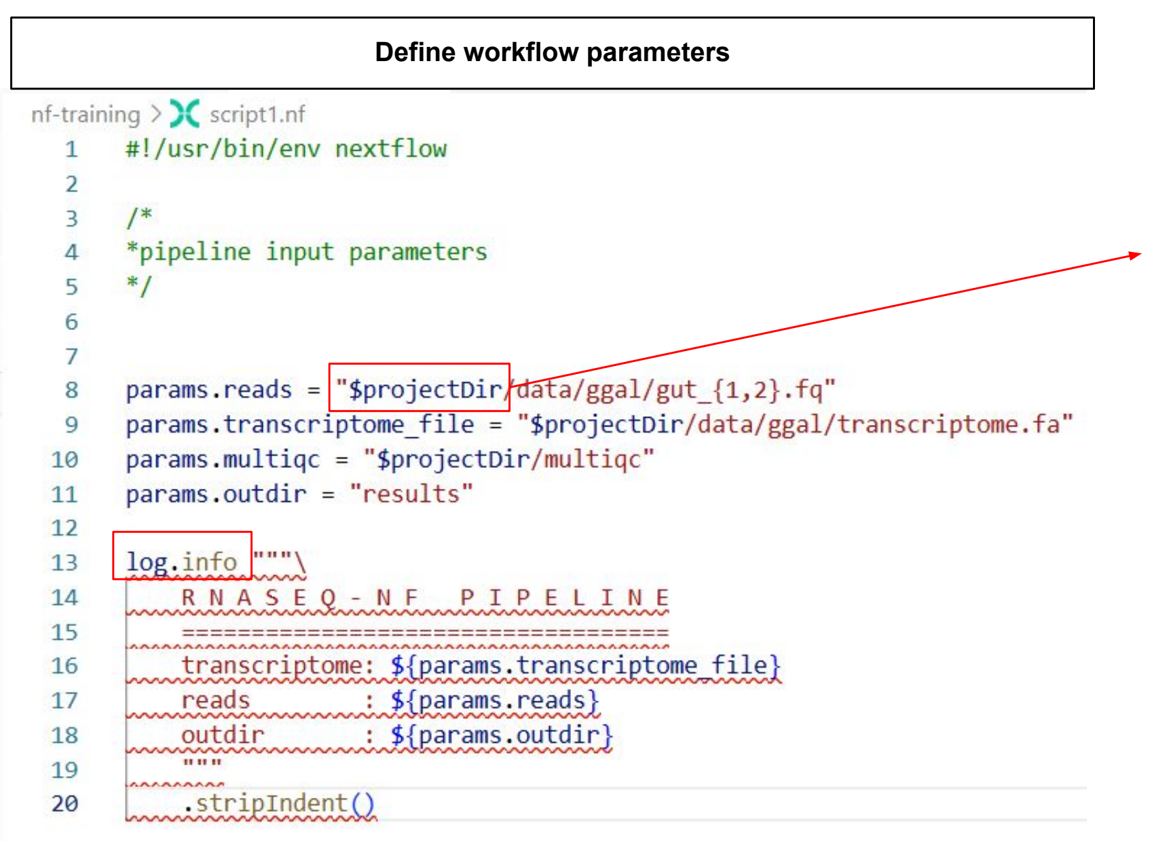
Scripts

script1.nf

a. Define workflow parameters
   i. Inputs
      1. $projectDir
      2. Fastq's
      3. transcriptome.fa
   ii. Outputs
      1. **Multiqc outdir**
      2. **Results outdir**
   iii. Notes
      1. log.info command outputs multiline message and saves copy of info into log execution file
      2. $projectDir is a dynamic variable whose value is determined at runtime when and where the workflow is executed
      3. $projectDir path = "/workspace/gitpod/nf-training"

Define workflow parameters

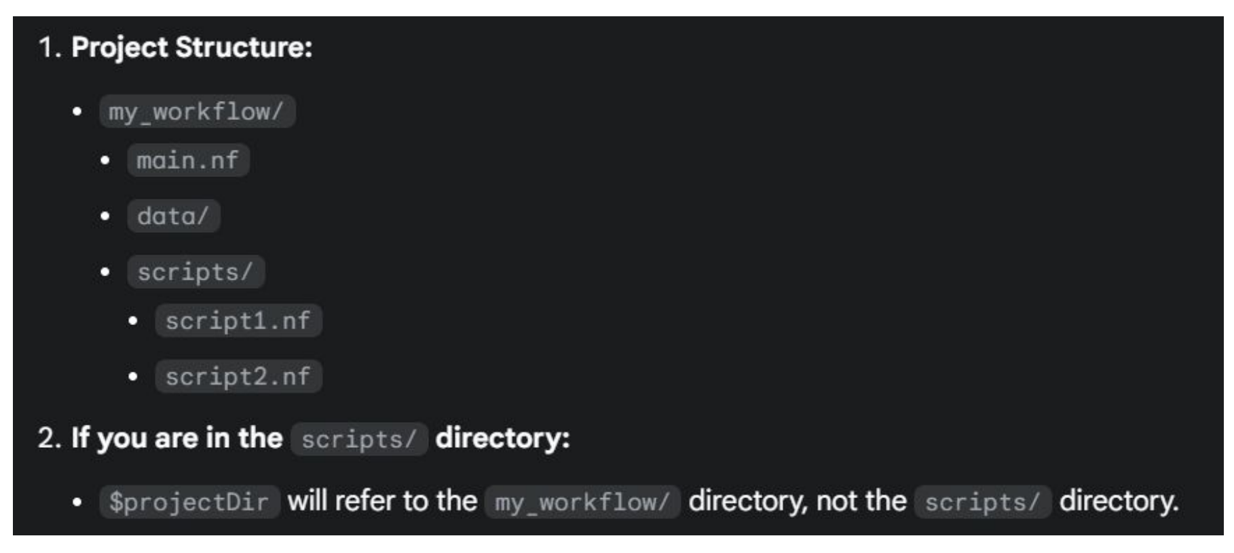




nf-training > script1.nf

```
#!/usr/bin/env nextflow

/*
*pipeline input parameters
*/


params.reads = "$projectDir/data/ggal/gut_{1,2}.fq"
params.transcriptome_file = "$projectDir/data/ggal/transcriptome.fa"
params.multiqc = "$projectDir/multiqc"
params.outdir = "results"

log.info """\
    R N A S E Q - N F   P I P E L I N E
    ===================================
    transcriptome: ${params.transcriptome_file}
    reads        : ${params.reads}
    outdir       : ${params.outdir}
    """
    .stripIndent()
```

Additional notes

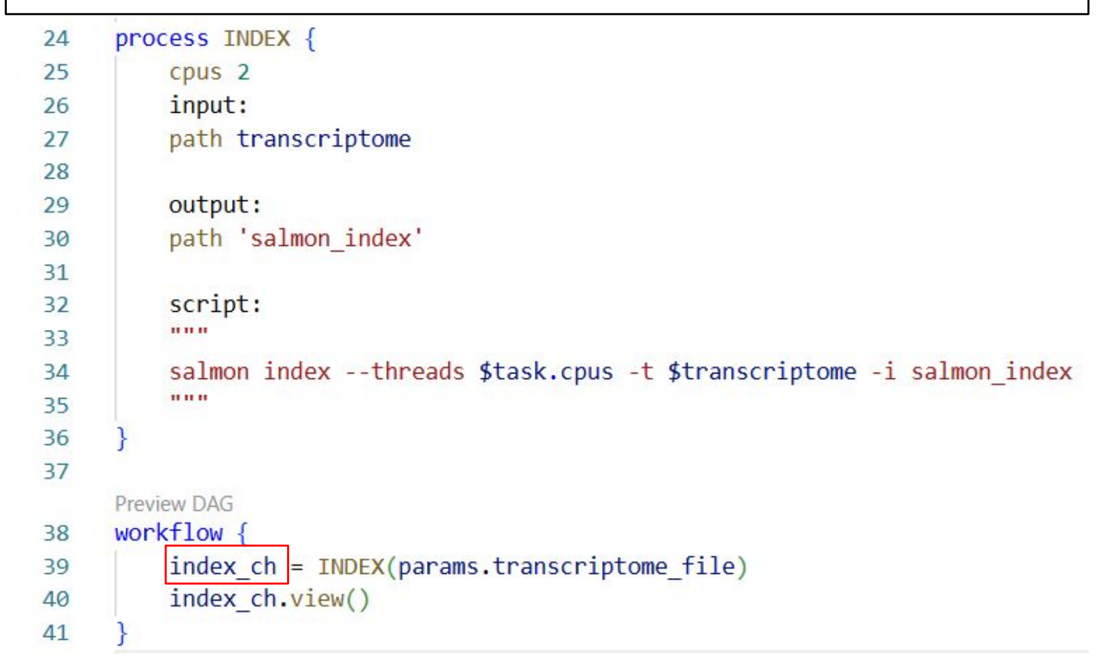


1. **Project Structure:**
   - `my_workflow/`
     - `main.nf`
     - `data/`
     - `scripts/`
       - `script1.nf`
       - `script2.nf`
2. **If you are in the `scripts/` directory:**
   - `$projectDir` will refer to the `my_workflow/` directory, not the `scripts/` directory.

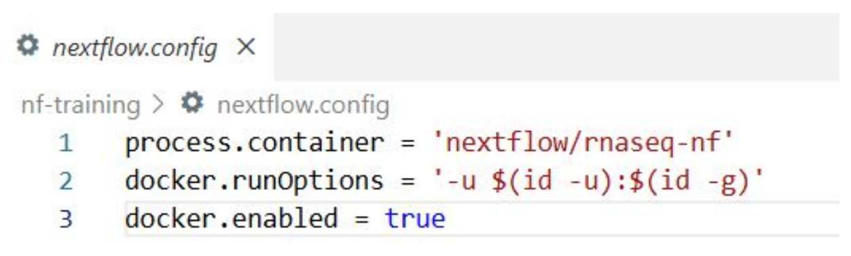
script2.nf

a. Create transcriptome index file
   i. Inputs
      1. transcriptome.fa path = params.transcriptome_file
   ii. Outputs
      1. Workflow scope containing index_ch channel
      2. Salmon_index outputs to index_ch #indexed transcriptome
   iii. Notes
      1. process INDEX() uses salmon
      2. Ensure nextflow.config defines salmon!!! See docker container image (line 1)
      3. Nextflow.config must be in $projectDir (currentdir)
      4. Ensure docker is enabled (line 3)

Process INDEX() & nextflow.config



```
process INDEX {
    cpus 2
    input:
    path transcriptome

    output:
    path 'salmon_index'

    script:
    """
    salmon index --threads $task.cpus -t $transcriptome -i salmon_index
    """
}

Preview DAG
workflow {
    index_ch = INDEX(params.transcriptome_file)
    index_ch.view()
}
```



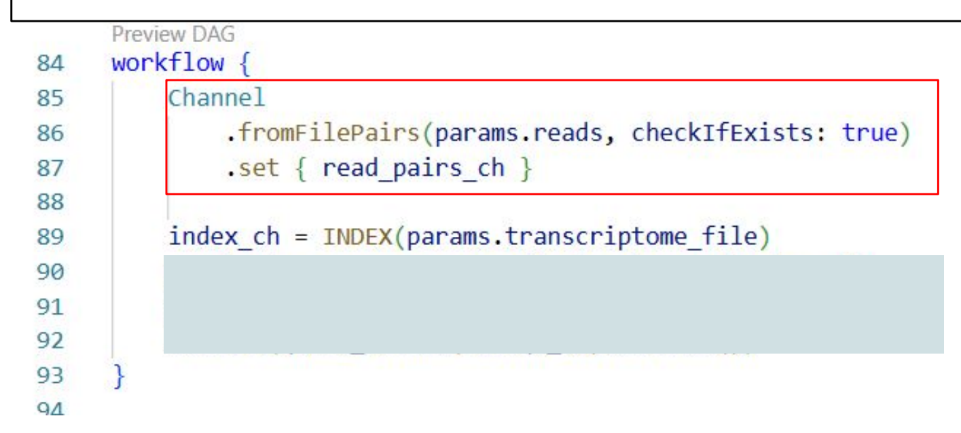
nextflow.config
nf-training > nextflow.config

```
1   process.container = 'nextflow/rnaseq-nf'
2   docker.runOptions = '-u $(id -u):$(id -g)'
3   docker.enabled = true
```

script3.nf

a. Collect read files by pairs into a channel
   i. Inputs
      1. params.reads #location of fq's
      2. fromFilePairs #known channel factory
   ii. Outputs
      1. read_pair_ch #a new channel variable
   iii. Notes
      1. fromFilePairs input is glob pattern and returns a channel of tuples. Each tuple contains two items: 1) read pair prefix 2) list of file paths
      2. .set ~ = , just used to assign
      3. Bonus points for wildcard function, but must be wrapped in single quotes if used

Collect paired fq files

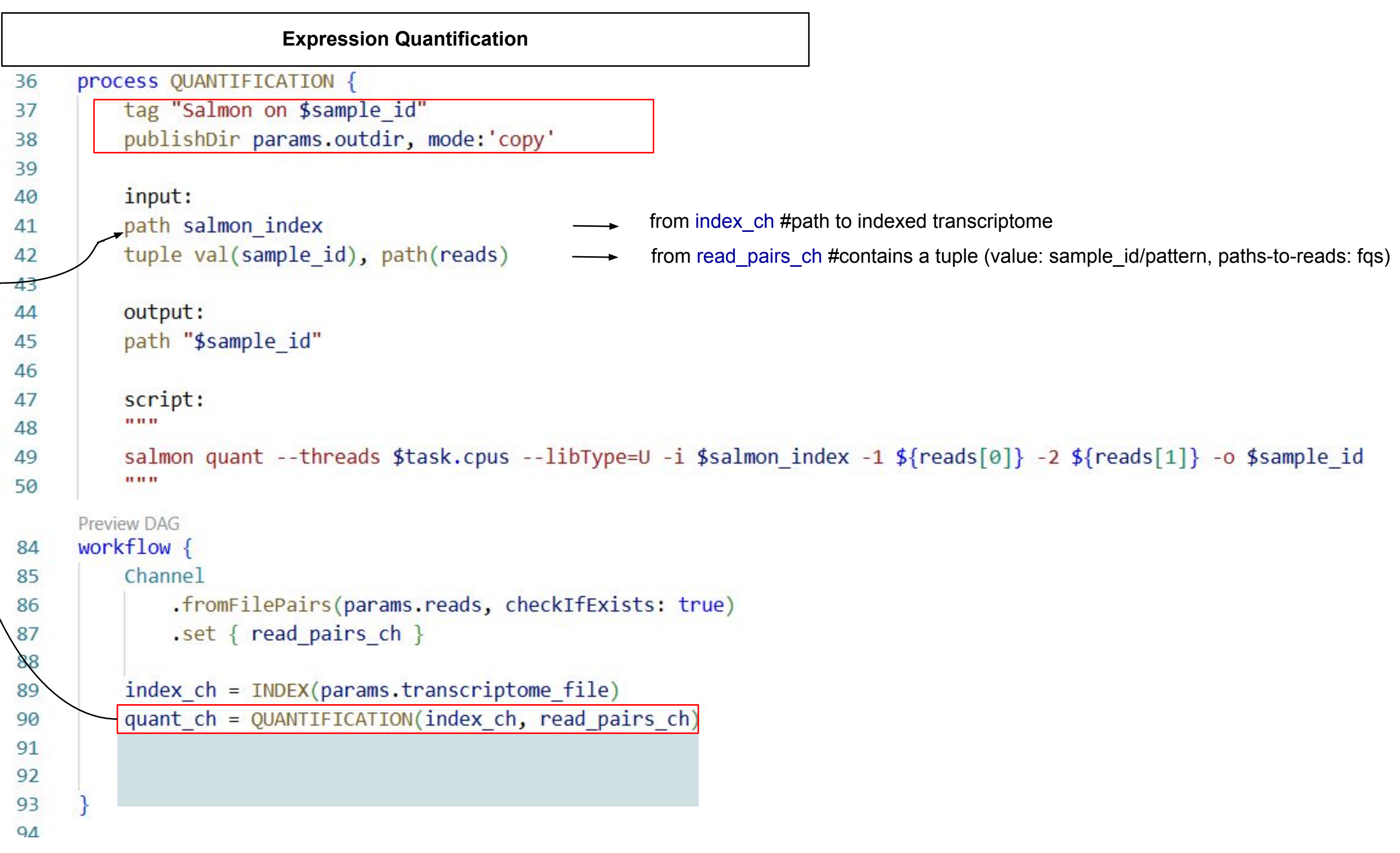


```
Preview DAG
workflow {
    Channel
        .fromFilePairs(params.reads, checkIfExists: true)
        .set { read_pairs_ch }

    index_ch = INDEX(params.transcriptome_file)



}
```

script4.nf

a. Expression quantification → with process QUANTIFICATION(read_pair_ch)
   i. Inputs
      1. Index_ch #previous output from process INDEX()
      2. read_pair_ch #previous output from known fromFilePairs channel
   ii. Outputs
      1. Workflow scope containing quant_ch channel
   iii. Notes
      1. Add tag to create more readable execution log
      2. Add publishDir to store the process results in outdir of choice (already defined as results/

Expression Quantification



```
process QUANTIFICATION {
    tag "Salmon on $sample_id"
    publishDir params.outdir, mode:'copy'

    input:
    path salmon_index
    tuple val(sample_id), path(reads)

    output:
    path "$sample_id"

    script:
    """
    salmon quant --threads $task.cpus --libType=U -i $salmon_index -1 ${reads[0]} -2 ${reads[1]} -o $sample_id
    """
Preview DAG
workflow {
    Channel
        .fromFilePairs(params.reads, checkIfExists: true)
        .set { read_pairs_ch }

    index_ch = INDEX(params.transcriptome_file)
    quant_ch = QUANTIFICATION(index_ch, read_pairs_ch)


}
```

path salmon_index → from index_ch #path to indexed transcriptome
tuple val(sample_id), path(reads) → from read_pairs_ch #contains a tuple (value: sample_id/pattern, paths-to-reads: fqs)

script5.nf

a. FASTQC
   i. Inputs
      1. read_pair_ch #previous output from known fromFilePairs channel
   ii. Outputs
      1. Fqc's into sample_id folders
      2. fastqc_ch channel #contains fqc paths
   iii. Notes
      1. process FASTQC() makes a fastqc folder for each sample

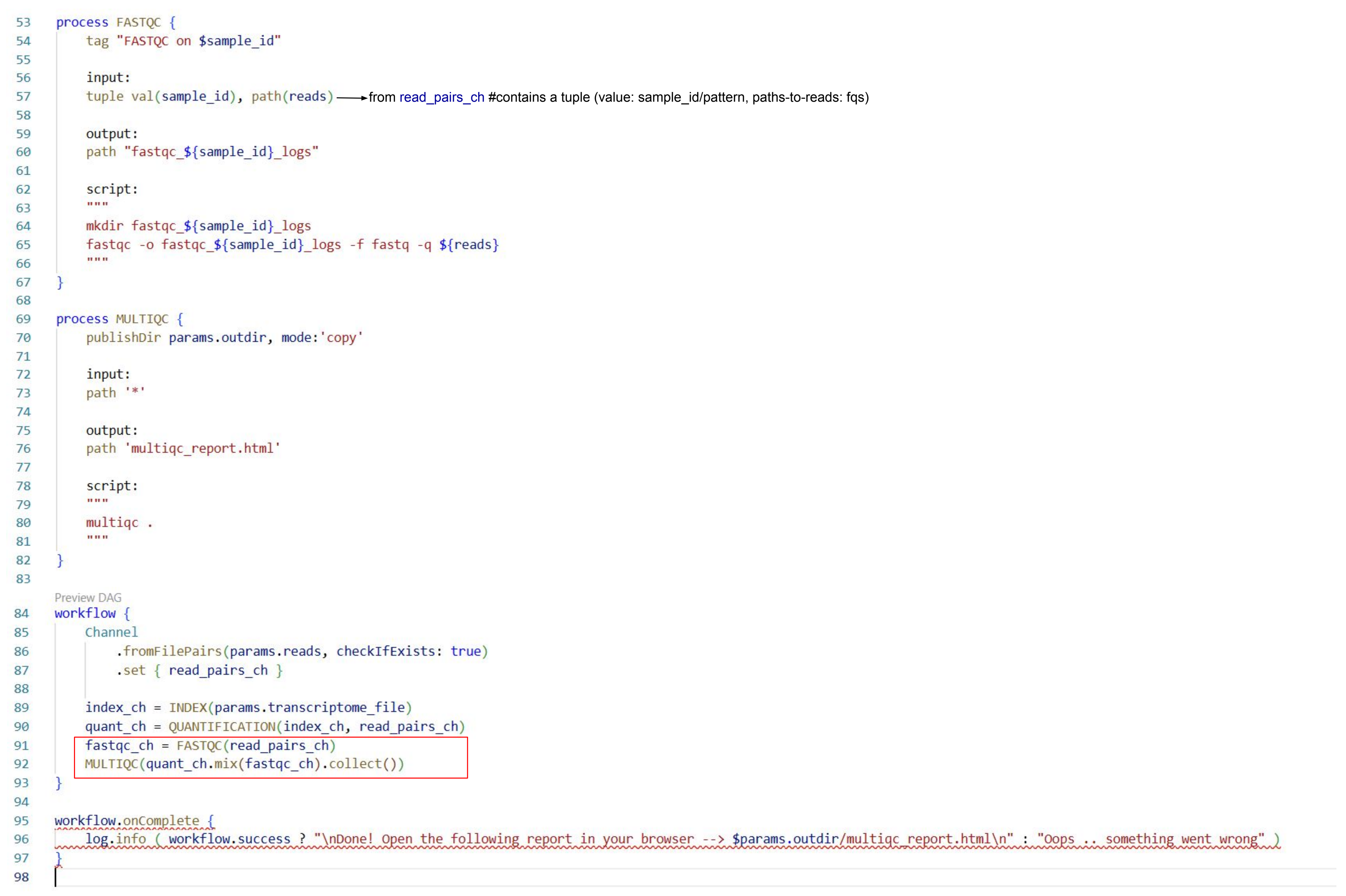
script6.nf

a. MultiQC report
   i. Inputs
      1. fastqc_ch channel #contains fqc paths
   ii. Outputs
      1. final report in results/ in current workdir
   iii. Notes
      1. process MULTIQC()
      2. mix and collect operators together gather the outputs of quant_ch and fastq_ch as single input to ensure return of complete channel connects as single element

script7.nf

a. Handle event completion
   i. Notes
      1. Execute action following workflow completion (workflow.onComplete)
      2. Asynchronous tasks
      3. Bonus points for adding email notification in nextflow.config

FASTQC, MULTIQC, Event completion



```
process FASTQC {
    tag "FASTQC on $sample_id"

    input:
    tuple val(sample_id), path(reads)

    output:
    path "fastqc_${sample_id}_logs"

    script:
    """
    mkdir fastqc_${sample_id}_logs
    fastqc -o fastqc_${sample_id}_logs -f fastq -q ${reads}
    """
}

process MULTIQC {
    publishDir params.outdir, mode:'copy'

    input:
    path '*'

    output:
    path 'multiqc_report.html'

    script:
    """
    multiqc .
    """
}

Preview DAG
workflow {
    Channel
        .fromFilePairs(params.reads, checkIfExists: true)
        .set { read_pairs_ch }

    index_ch = INDEX(params.transcriptome_file)
    quant_ch = QUANTIFICATION(index_ch, read_pairs_ch)
    fastqc_ch = FASTQC(read_pairs_ch)
    MULTIQC(quant_ch.mix(fastqc_ch).collect())
}

workflow.onComplete {
    log.info ( workflow.success ? "\nDone! Open the following report in your browser --> $params.outdir/multiqc_report.html\n" : "Oops .. something went wrong" )
}
```

tuple val(sample_id), path(reads) → from read_pairs_ch #contains a tuple (value: sample_id/pattern, paths-to-reads: fqs)