Example RNA-seq Nextflow script from video

```
#!/usr/bin/env nextflow
// Defining necessary variables including the path to the reference genome and fastq files
params.ref = "/hpc/faculty/azeezoe/rna_seq/refs/22.fa"
params.reads = "/hpc/faculty/azeezoe/rna_seq/reads/*_R{1,2}.fq.gz"
params.outdir = "./results"
params.PE = true

// Fastqc process, creates zip files used for multiqc
process FASTQC {
    publishDir "${params.outdir}/fastqc", mode: 'copy'

    // Input are the reads channel
    input:
    tuple val(sample), val(extension), path(reads)

    output:
    path("*.zip")

    script:
    """
    fastqc ${reads}
    """
}

// FASTP_PE, fastp processing for paired end files. if params.PE (top) set to false then this process is not called
process FASTP_PE {
    publishDir "${params.outdir}/fastp", mode: 'copy'

    input:
    tuple val(sample), val(extension), path(reads)

    output:
    // Defining channel output emitted as "reads", and the fastp.json report as "json"
```

```
    tuple val(sample), val(extension),
path("${sample}_{1,2}.trimmed.{${extension[0]},${extension[1]}}"), emit: reads
    path("${sample}.fastp.json"), emit: json

    script:
    // Splitting the reads and extension lists into individual variables
    def read1 = reads[0]
    def read2 = reads[1]
    def extension1 = extension[0]
    def extension2 = extension[1]

    """
    fastp -i ${read1} -I ${read2} -o ${sample}_1.trimmed.${extension1} -O
${sample}_2.trimmed.${extension2} -j ${sample}.fastp.json
    """
}

// Multiqc allows for easy visualization of fastp and fastqc reports
process MULTIQC{
    publishDir "${params.outdir}/multiqc", mode: 'copy'

    input:
    // takes the fastp.json output and all the zip files from fastqc
    path(fastp_out)
    path(fastqc_out)

    output:
    path('*')

    script:
    // "multiqc ." effectively means "run multiqc on everything in the entire directory you
are in".
    // Because we aren't "in" any directory as we have to specify inputs to a process,
    // we will need to use the ".collect()" method for inputs when we call this process in the
workflow block (see below),
    // so as to hand multiqc everything we need at once.
    """
    multiqc .
    """

}
```

```
// Indexing our transcriptome
process SALMON_IDX {
    input:
    path(transcriptome)

    output:
    path("salmon_idx")

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

// Salmon quantification step, requires our trimmed reads and the salmon index
process SALMON_QUANT_PE {
    publishDir "${params.outdir}/salmon", mode: 'copy'

    input:
    path(salmon_idx)
    tuple val(sample), val(extension), path(reads)

    output:
    path("*")

    script:
    def read1 = reads[0]
    def read2 = reads[1]

    """
    salmon quant --threads ${task.cpus} -i ${salmon_idx} -l A -1 ${read1} -2 ${read2} -o
${sample}.quant
    """
}



workflow {
    // Handles .fastq, .fq, .fastq.gz, .fq.gz files automatically, no need to modify
    if ( params.PE ) {
        reads_ch = channel.fromFilePairs( params.reads, checkIfExists: true )
```

```
        .map { sample_id, files ->
            def extensions = files.collect { file ->
                def file_name = file.name
                def idx = file_name.indexOf(".")
                def extension = file_name.substring(idx+1)
                extension
                }
            tuple(sample_id, extensions, files)
        }
    }
    else {
        reads = channel.fromPath( params.reads, checkIfExists: true )
        reads_ch = reads.map { file ->
            def file_name = file.name
            def idx = file_name.indexOf(".")
            def sample_id = file_name.substring(0, idx)
            def extension = file_name.substring(idx+1)
            tuple( sample_id, extension, file )
        }
    }

    // Viewing what the reads channel looks like
    reads_ch.view()

    // Fastqc and fastp takes reads channel input
    fastqc_out = FASTQC(reads_ch)
    fastp_out = FASTP_PE(reads_ch)

    // Multiqc requires just the "json" output from fastp, and everything from fastqc.
    // By default nextflow passes inputs as a stream to a process, but in this case we need
    // to hand everything to the multiqc process at once. To do this we use the ".collect()"
method
    MULTIQC(fastp_out.json.collect(), fastqc_out.collect())

    salmon_idx = SALMON_IDX(params.ref)
    // Notice again that we specify we only want to supply the reads output from fastp to
salmon quant
    // To do this we use "dot notation", fastp_out.reads (which we defined with the "emit"
keyword in the process block)
    SALMON_QUANT_PE(salmon_idx, fastp_out.reads)
}
```

Revision #2
Created 2025-02-14 16:12:54 UTC by Admin
Updated 2025-02-14 16:14:39 UTC by Admin