

Building Pipelines in DolphinNext



Onur Yukselen, Alper Kucukural
Bootcamp, May 05, 2021



Content

- ▶ Pipeline Basics
 - Process, Inputs, Outputs, Script
- ▶ Building Blocks
 - Val, File, Set
- ▶ How it Works?

Pipeline Basics

The screenshot shows the Biocore DolphinNext BETA web interface. At the top, there is a navigation bar with links for 'Projects', 'Pipelines', 'Amazon', 'Profiles', and a 'BETA VERSION' indicator. On the left, a sidebar displays the user 'Onur Yükselen' (Online) and a search bar. Below the search bar is a 'PIPLINES' section with links to 'Public Pipelines', 'Single Cell', 'piPipes', 'Robs', and 'STAR'. The main content area is titled 'Biocore Generation' and 'Public Pipelines'. It lists six pipelines:

- Single Cell-10X Genomics**: This pipeline maps 10X Genomics reads to selected genome (by using Tophat2), measures digital gene expression (by using ESAT) and finally creates UMI distributions table for expression analysis. Inputs: "Determined_fastq": In order to enter reads that are shown as an example at below, followin
- Single Cell-Indrop**: This pipeline maps inDrop reads to selected genome (by using Tophat2), measures digital gene expression (by using ESAT) and finally creates UMI distributions table for expression analysis. Inputs: "Determined_fastq": output of Bcl2fastq software which has a structure as shown at below. In order t
- RSEM Pipeline**: Tophat pipeline includes Quality Control, rRNA filtering, Genome Alignment using Tophat. Steps: 1) For Quality Control, we use FastQC to create qc outputs. 2) Bowtie2 is used to filtered out rRNA reads. 3) TopHat is a program that aligns RNA-Seq reads to a genome in order to identify exon-exon sp
- HISAT2 pipeline**: Tophat pipeline includes Quality Control, rRNA filtering, Genome Alignment using Tophat. Steps: 1) For Quality Control, we use FastQC to create qc outputs. 2) Bowtie2 is used to filtered out rRNA reads. 3) TopHat is a program that aligns RNA-Seq reads to a genome in order to identify exon-exon sp
- STAR Pipeline**: Tophat pipeline includes Quality Control, rRNA filtering, Genome Alignment using Tophat. Steps: 1) For Quality Control, we use FastQC to create qc outputs. 2) Bowtie2 is used to filtered out rRNA reads. 3) TopHat is a program that aligns RNA-Seq reads to a genome in order to identify exon-exon sp
- Tophat2 Pipeline**: Tophat pipeline includes Quality Control, rRNA filtering, Genome Alignment using Tophat. Steps: 1) For Quality Control, we use FastQC to create qc outputs. 2) Bowtie2 is used to filtered out rRNA reads. 3) TopHat is a program that aligns RNA-Seq reads to a genome in order to identify exon-exon sp

Each pipeline card has a 'LEARN MORE' button. A red circle highlights the 'LEARN MORE' button for the 'Single Cell-Indrop' pipeline. At the bottom, there is a page navigation bar with links for '1', '2', '3', and 'next'.

Pipeline Basics – Input/Output Parameter

Screenshot of the Biocore DolphinNext BETA pipeline generation interface.

Inputs:

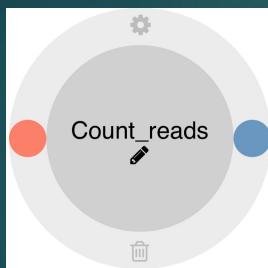
Given Name	Select Items
cellBarcodeFile	/project/umw_garberlab/yukseleo/barcode.txt
determined_fastq	GSE109033
mate_split	single
cutoff_reads_for_valid_cell	3000
genomeTypePipeline	mousetest_m

Pipeline Graph:

Run 4 - Report:

PROCESS	PUBLISHED DIRECTORY	VIEW FORMAT
Run Notes	Description	Markdown
Multiqc	Multiqc	HTML
Overall Summary	Summary	Table

Pipeline Basics – Inputs/Outputs/Script



inputs outputs
script

Test Server DolphinNext Pipelines Projects Run Status

Onur Yüksel Online

Created by onuryuksele on 2020-04-28 02:10:24 • Last edited on 2020-04-28 16:26:05

Revision: 1

Edit/Delete Process

Name: Count_reads

Description:

Menu Group: Tutorial

Parameters:

Inputs	Input Name	Operators	Operator Content	Optional
reads (fastq, file)	read	x	x	
Add input...				

Outputs	Output Name	Operators	Operator Content	Optional	Regular Expression
txtFile (txt, file)	".*.count.txt"	x	x		
Add input...					

Script:

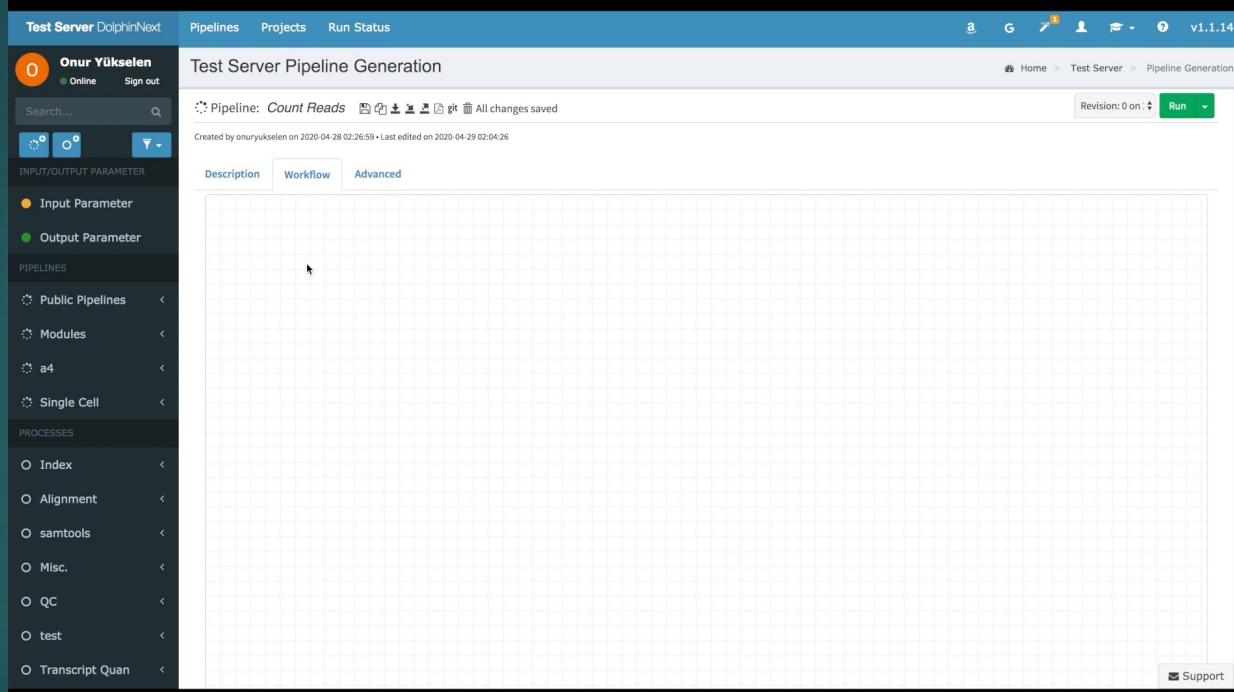
```
1 wc -l $read | awk '{print \$0/4}' > ${read}.count.txt
```

nextflow

```
if (!params.reads){params.reads = ""}  
g_7_reads_g_12 = file(params.reads, type: 'any')  
process Count_reads {  
  
    publishDir params.outdir, overwrite: true, mode: 'copy',  
    saveAs: {filename -> if (filename == './*.count.txt$')  
        "count/${filename}"}  
  
    input:  
        file read from g_7_reads_g_12  
  
    output:  
        file "*.count.txt" into g_12_txtfile  
  
    script:  
        ""  
        wc -l $read | awk '{print \$0/4}' > ${read}.count.txt  
        ""  
}
```

Support

Pipeline Basics – Drag&Drop



Building Blocks



Qualifier

Val

Examples

"mouse"



File

exp_rep1.fasta



Set

[“exp_rep1”, e

Input Parameters i

Inputs i

outputs (fastq, file) i

Add input...

Output Name i

Output Parameters i

Outputs i

outputHTML (html, file) i

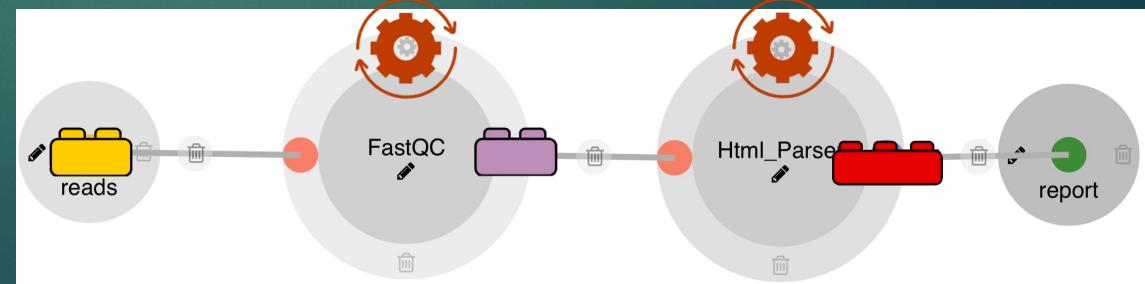
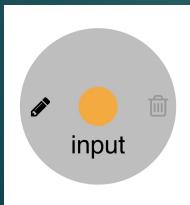
Add input...

Output Name i

Script

```
1 fastqc ${reads}
```

Input Parameters



Adding New Parameter

The screenshot shows the DolphinNext software interface with a dark theme. On the left, there's a sidebar with sections for INPUT/OUTPUT PARAMETERS, PIPELINES, and PROCESSES. The PIPELINES section has a 'Tutorial' pipeline expanded, showing a 'fastqc' step. The PROCESSES section lists 'Misc.', 'QC', and 'Tutorial' with a 'FastQC' step under 'Tutorial'. A central modal window titled 'Add New Parameter' is open, containing fields for Name (FastQC), Identifier, Qualifier (file), and File Type. Below this, there are tabs for Parameters, Input Parameters, Output Parameters, and Script. The Input Parameters tab shows an input named 'read' with a dropdown set to 'reads (fasta, file)'. The Output Parameters tab shows an output named '.html' with a dropdown set to 'outputHTML (html, file)'. The Script tab contains the command:

```
1 fastqc ${reads}
```

. At the bottom right of the modal is a 'Save changes' button.

Adding New Parameter

Adding New Parameter



Identifier ⓘ	inputText
Qualifier	val

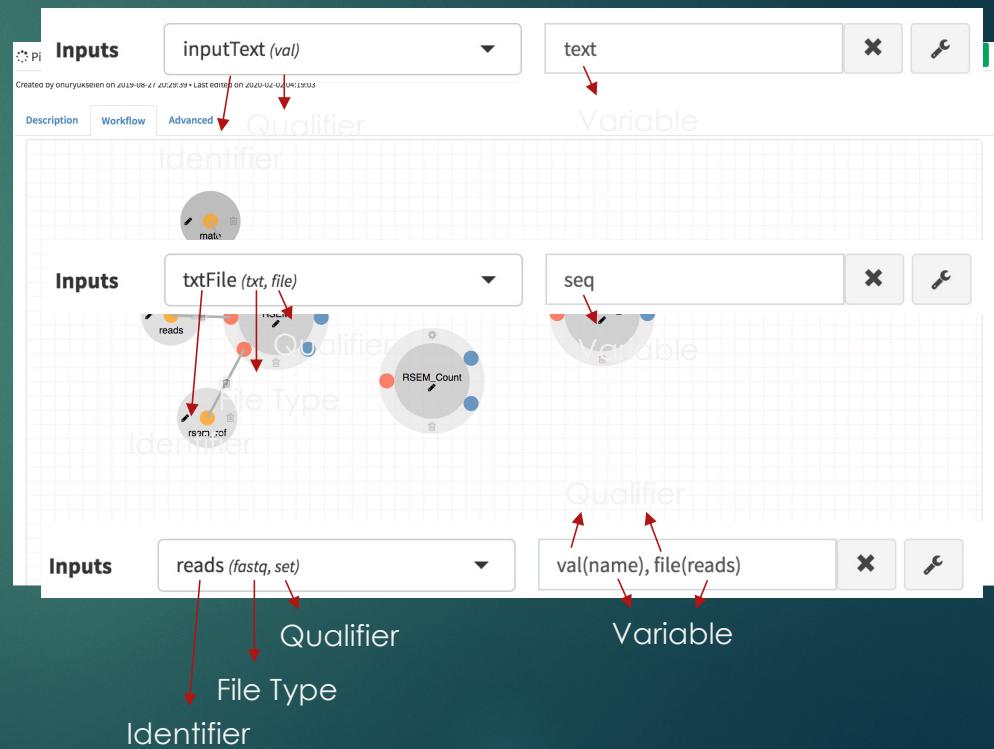


Identifier ⓘ	txtFile
Qualifier	file
File Type ⓘ	txt



Identifier ⓘ	reads
Qualifier	set
File Type ⓘ	fastq

Using Parameter in the Process

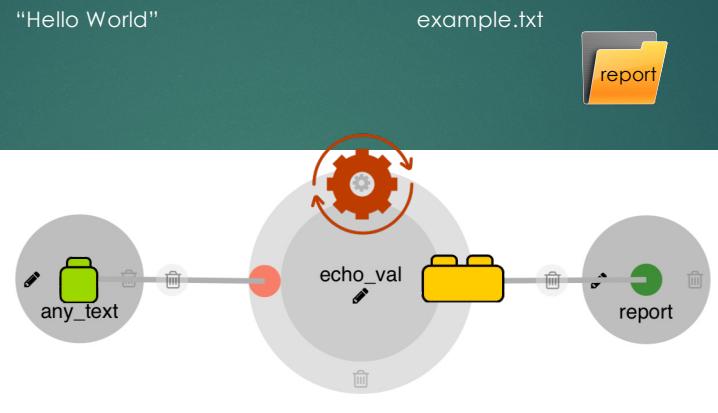


How it Works? (1) – Using Val

User Inputs

Inputs

Given Name	Select Items
~ User Inputs ~	
any_text	Hello World!



Run Logs

```
NEXTFLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run1/run422/nextflow.nf` [84/2dfda8] Submitted process > echo_val
```

Inputs text

Script

```
1 echo $text > example.txt
```

\$ echo Hello World! > example.txt

Outputs "example.txt"

Run Reports

Echo Val

example.txt

Hello World!

How it Works? (2) – Using File

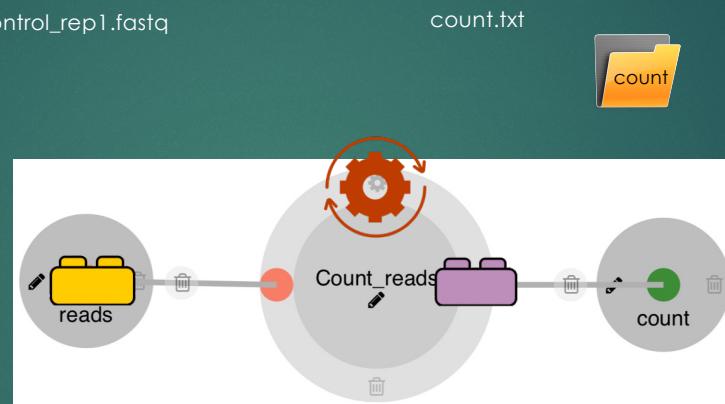
Run Inputs

Inputs

Given Name Select Items

~ User Inputs ~

reads /share/data/umw_biocore/dnext_data/tutorial/fastq_data/single/control_rep1.fastq  



Inputs reads (fastq, file) read  

Script

```
1 wc -l $read | awk '{print \$0/4}' > count.txt
```

```
$ wc -l control_rep1.fastq | awk '{print $0/4}' > count.txt
```

Outputs txtFile (txt, file) "count.txt"  

Run Logs

```
NEXTFLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run2/run423/nextflow.nf
[8b/62795a] Submitted process > Count_reads
```

Run Reports

Count Reads
count.txt 24288

How it Works? (3) – Using File – Multiple Input Problem?

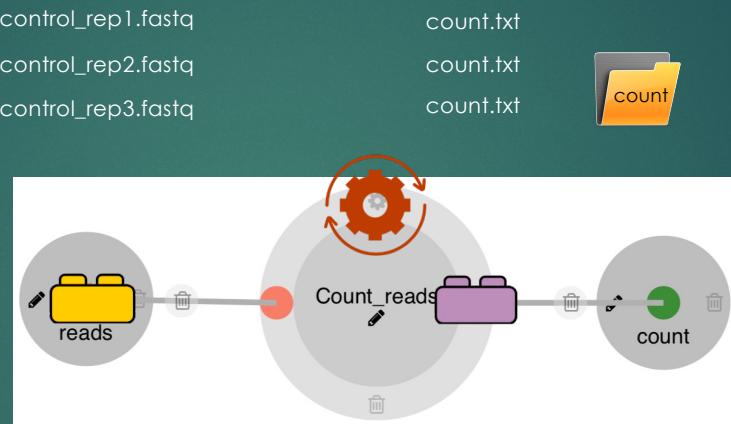
Run Inputs

Inputs

Given Name Select Items

~ User Inputs ~

reads 3 single-end reads



Run Logs

```
NEXTFLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run2/run423/nextflow.nf`
[5f/19a1f0] Submitted process > Count_reads (1)
[56/5a17f6] Submitted process > Count_reads (2)
[d9/7557e2] Submitted process > Count_reads (3)
```

Inputs reads (fasta, file) read

Script 1 wc -l \$read | awk '{print \\$0/4}' > count.txt

Outputs txtFile (txt, file) "count.txt"

Run Reports

Count Reads

count.txt 17568

How it Works? (3) – Using File - Problem Fixed

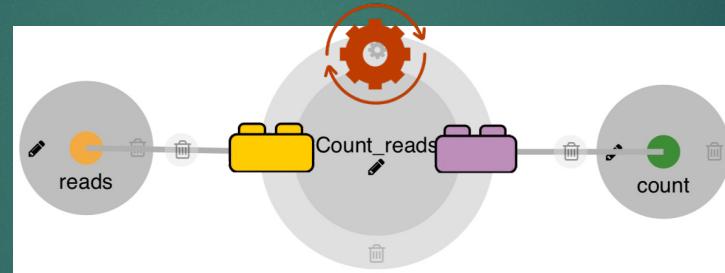
Run Inputs

Inputs

Given Name Select Items

~ User Inputs ~

reads 3 single-end reads



Inputs reads (fastq, file) read

Script

```
1 wc -l $read | awk '{print $0/4}' > ${read}.count.txt  
$read == ${read}
```

\$ wc -l control_rep1.fastq | awk '{print \$0/4}' > control_rep1.fastq.count.txt

Outputs txtFile (txt, file) *.count.txt

Run Logs

```
NEXTFLOW ~ version 20.01.0  
Launching `/home/oy28w/tutorials/run2/run423/nextflow.nf`  
[5f/19a1f0] Submitted process > Count_reads (1)  
[56/5a17f6] Submitted process > Count_reads (2)  
[d9/7557e2] Submitted process > Count_reads (3)
```

Run Reports

Count Reads	24288
control_rep1.fastq.count.txt	
control_rep2.fastq.count.txt	
control_rep3.fastq.count.txt	

How it Works? (4) – Using Set

Run Inputs

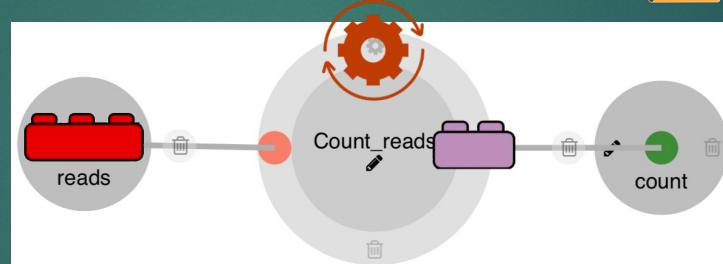
Inputs

Given Name Select Items

~ User Inputs ~

reads 3 single-end reads

["control_rep1", control_rep1.fasta]
["control_rep2", control_rep2.fasta]
["control_rep3", control_rep3.fasta]



control_rep1.txt
control_rep2.txt
control_rep3.txt



Run Logs

```
NEXTFLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run2/run423/nextflow.nf`
[5f/19a1f0] Submitted process > Count_reads (1)
[56/5a17f6] Submitted process > Count_reads (2)
[d9/7557e2] Submitted process > Count_reads (3)
```

Run Reports

Inputs reads (fastq, set) val(name), file(read)

Script

```
1 wc -l $read | awk '{print \$0/4}' > ${name}.txt
```

Outputs txtFile (txt, file) "\${name}.txt"

Count Reads	control_rep1.txt	24288
	control_rep2.txt	
	control_rep3.txt	

How it Works? (5) – Using Set for Paired Reads

```
["control_rep1", control_rep1.R1.fastq, control_rep1.R2.fastq]  
["control_rep2", control_rep2.R1.fastq, control_rep2.R2.fastq]  
["control_rep3", control_rep3.R1.fastq, control_rep3.R2.fastq]
```

Run Inputs

~ User Inputs ~

mate	pair
reads	3 paired-end reads

"pair"

```
control_rep1.txt  
control_rep2.txt  
control_rep3.txt
```



The diagram shows a Nextflow process named 'Count_reads'. It has two inputs: 'reads' (represented by a red block icon) and 'mate' (represented by a green block icon). The 'reads' input connects to a grey circle labeled 'Count_reads'. The 'mate' input connects to another grey circle labeled 'Count_reads'. A purple block icon (representing an output file) is connected to the 'Count_reads' circle. Below the process, there is a 'Script' section containing the command: \$ wc -l \$read | awk '{print \$0/4}' > \${name}.txt. At the bottom, there is an 'Outputs' section where the output is specified as txtFile (txt, file) and the name is \${name}.txt.

```
Inputs: reads (fastq, set), mate (file)  
Script:  
$ wc -l $read | awk '{print $0/4}' > ${name}.txt  
$ wc -l control_rep1.R1.fastq control_rep1.R2.fastq  
97152 control_rep1.R1.fastq  
97152 control_rep1.R2.fastq  
194304 total  
$ wc -l control_rep1.R1.fastq control_rep1.R2.fastq | awk '{print $0/4}'> ${name}.txt  
24288  
24288  
48576  
Outputs: txtFile (txt, file), "${name}.txt"
```

Run Logs

```
NEXTFLOW ~ version 20.01.0  
Launching `/home/oy28w/tutorials/run4/run425/nextflow.nf`  
[a0/bfffa5] Submitted process > Count_reads (2)  
[8e/c6a0e] Submitted process > Count_reads (1)  
[a8/b5f2d6] Submitted process > Count_reads (3)
```

Run Reports

Count Reads
control_rep1.txt
control_rep2.txt
control_rep3.txt

24288
24288
48576

How it Works? (6) – Using Set for Single & Paired Reads

Run Inputs

~ User Inputs ~

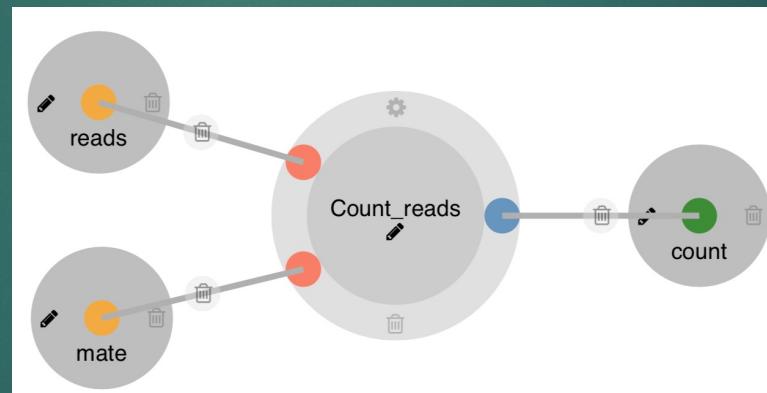
mate	pair
reads	3 paired-end reads

Run Logs

NEXT FLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run4/run425/nextflow.nf`
[a0/bfffa5] Submitted process > Count_reads (2)
[8e/6c6a0e] Submitted process > Count_reads (1)
[a8/b5f2d6] Submitted process > Count_reads (3)

Run Reports

Count Reads	
control_rep1.txt	control_rep1.R1.fastq 24288
control_rep2.txt	control_rep1.R2.fastq 24288
control_rep3.txt	



Run Inputs

~ User Inputs ~

mate	single
reads	3 single-end reads

Run Logs

NEXT FLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run4/run425/nextflow.nf`
[a0/bfffa5] Submitted process > Count_reads (2)
[8e/6c6a0e] Submitted process > Count_reads (1)
[a8/b5f2d6] Submitted process > Count_reads (3)

Run Reports

Count Reads	
control_rep1.txt	control_rep1.fastq 24288
control_rep2.txt	
control_rep3.txt	

Inputs: reads (fastq, set) val(name), file(read)

Script:

```
1 if [ "${mate}" == "pair" ]; then
2     wc -l $read | awk '{print $2 "\t" $1/4}' | head -2 > ${name}.txt
3 else
4     wc -l $read | awk '{print $2 "\t" $1/4}' > ${name}.txt
5 fi
```

Outputs: txtFile (txt, file) "\${name}.txt"

How it Works? (7) – Merging tables

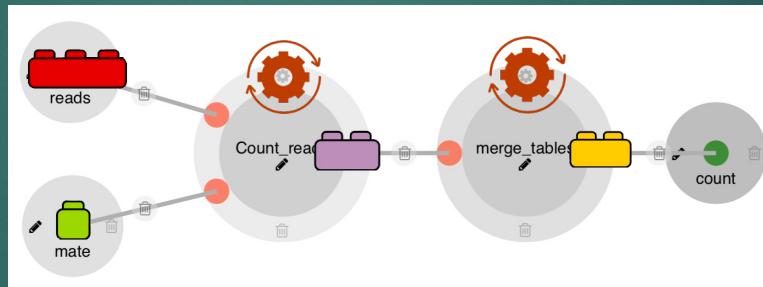
["control_rep1", control_rep1.R1.fastq, control_rep1.R2.fastq]
["control_rep2", control_rep2.R1.fastq, control_rep2.R2.fastq]
["control_rep3", control_rep3.R1.fastq, control_rep3.R2.fastq]

"pair"

Run Inputs

~ User Inputs ~

mate	pair
reads	3 paired-end reads



Inputs

Script

```
1 echo -e 'filename\tcount' > counts.tsv
2 cat $txtFiles >> counts.tsv
```

\$ cat control_rep1.txt control_rep2.txt control_rep3.txt >> counts.tsv

Outputs

NEXTFLOW ~ version 20.01.0
Launching `/home/oy28w/tutorials/run7/run428/nextflow.nf`
[db/ef1138] Submitted process > Count_reads (1)
[3c/6ae923] Submitted process > Count_reads (2)
[d8/cbb4b6] Submitted process > Count_reads (3)
[83/8a6743] Submitted process > merge_tables

Run Reports

filename	count
control_rep1.R1.fastq	24288
control_rep1.R2.fastq	24288
control_rep2.R1.fastq	15693
control_rep2.R2.fastq	15693
control_rep3.R1.fastq	17008
control_rep3.R2.fastq	17008

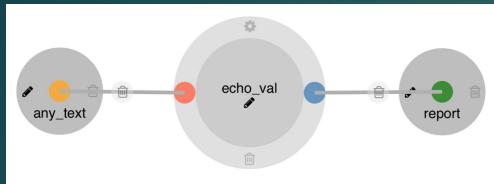
Operators

- ▶ Reference Documentation for Operators:
 - <https://www.nextflow.io/docs/latest/index.html>

The screenshot shows the Nextflow documentation website. The header includes a logo, a search bar, and a navigation menu with links like 'Get started', 'Basic concepts', 'Nextflow scripting', 'Processes', 'Channels', and 'Operators'. The 'Operators' link is highlighted. The main content area has a breadcrumb trail 'Docs » Operators' and a section titled 'Operators'. It describes what operators are and lists seven groups: Filtering operators, Transforming operators, Splitting operators, Combining operators, Forking operators, Maths operators, and Other operators. The 'nextflow' logo is at the bottom.

Summary

(1) Val



Inputs: inputText (val) → text

Outputs: txtFile (txt, file) → "example.txt"

Script

```

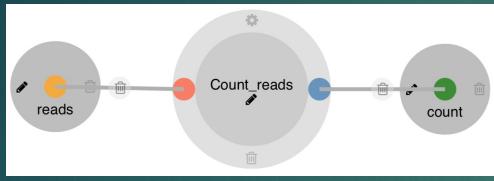
1 echo $text > example.txt
  
```

Echo Val

example.txt

Hello World!

(3) File



Inputs: reads (fastq, file) → read

Outputs: txtFile (txt, file) → ".count.txt"

Script

```

1 wc -l $read | awk '{print $0/4}' > ${read}.count.txt
  
```

Count Reads

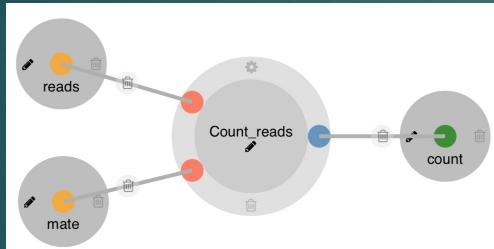
control_rep1.fastq.count.txt

control_rep2.fastq.count.txt

control_rep3.fastq.count.txt

24288

(6) Set



Inputs: reads (fastq, set) → val(name), file(read)

Outputs: txtFile (txt, file) → "\${name}.txt"

Script

```

1 if [ "${mate}" == "pair" ]; then
2   wc -l $read | awk '{print $2 "\t" $1/4}' | head -2 > ${name}.txt
3 else
4   wc -l $read | awk '{print $2 "\t" $1/4}' > ${name}.txt
5 fi
  
```

Count Reads

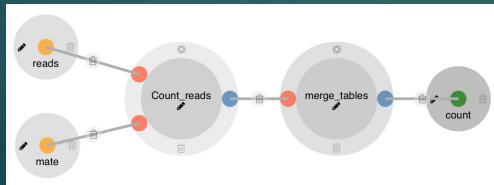
control_rep1.txt

control_rep2.txt

control_rep3.txt

control_rep1.R1.fastq 24288
control_rep1.R2.fastq 24288
control_rep2.R1.fastq 15693
control_rep2.R2.fastq 15693
control_rep3.R1.fastq 17008
control_rep3.R2.fastq 17008

(7) Operator



Inputs: txtFile (txt, file) → txtFiles

Outputs: outFileTSV (tsv, file) → "counts.tsv"

Script

```

1 echo -e 'filename\tcount' > counts.tsv
2 cat $txtFiles >> counts.tsv
  
```

filename	count
control_rep1.R1.fastq	24288
control_rep1.R2.fastq	24288
control_rep2.R1.fastq	15693
control_rep2.R2.fastq	15693
control_rep3.R1.fastq	17008
control_rep3.R2.fastq	17008

Next Step: Building Pipeline

Overview

- Before you start
- Getting Started
- Exercise 1: Creating processes
 - FastQC process
 - Hisat2 process
 - RSeQC process
- Exercise 2: Building a pipeline
- Exercise 3: Running a pipeline
- Exercise 4 (optional): Supporting single and paired end reads

If you prefer, you can click on the video links to follow the tutorial in a video.

The screenshot shows the 'Test Server Pipeline Generation' interface. On the left, there's a sidebar with user information ('Onur Yıldızlıan') and sections for 'Input Parameters', 'Output Parameters', 'PROCESSES' (with 'Tutorial' expanded to show 'FastQC', 'Hisat2', 'RSeQC'), and 'PIPELINES'. The main area is titled 'Test Server Pipeline Generation' and shows a 'Workflow' diagram on a grid. The workflow consists of several nodes: 'Input_R1...' (orange), 'FastQC...' (grey), 'FastQC...' (grey), 'RSeQC...' (grey), 'bedfile' (grey), and 'Hisat2...' (grey). Arrows indicate the flow from 'Input_R1...' to the first 'FastQC...' node, then to the second 'FastQC...' node, then to 'RSeQC...', then to 'bedfile', and finally to the two 'Hisat2...' nodes. The 'Workflow' tab is selected at the top of the diagram area. A progress bar at the bottom right indicates 'Exercise 2 Building a pipeline' is 5:37 / 8:57 completed.