Example pipeline: fastq to bam

Sahil Seth

2015-11-20

Contents

You may visit docs.flowr.space for more details.

If you face any issues, please feel free to raise a issue on github.

1 Setup up flowr

Requirements:

- R version > 3.1, preferred 3.2
- install flowr from sahilseth.github.io/drat, provides a more recent version than CRAN.

```
#install.packages("params", repos = "http://cran.rstudio.com")
## for a latest stable version (updated every few days):
install.packages("flowr", repos = c(CRAN = "http://cran.rstudio.com",
                                    DRAT = "http://sahilseth.github.io/drat"))
```

After installation run setup(), this will copy the flowr's helper script to ~/bin. Please make sure that this folder is in your $PATH variable.

```
library(flowr)
setup()
```

```
## Warning in file.copy(confs, flow_conf_path, overwrite = FALSE): problem
## copying /Library/Frameworks/R.framework/Versions/3.2/Resources/library/
## ngsflows/conf/ngsflows.conf to /Users/sahilseth/flowr/conf/ngsflows.conf:
## No such file or directory
```

```
##
## Attaching package: 'knitr'
##
## The following object is masked from 'package:params':
##
##     kable
```

Then we need to test whether we are able to submit jobs to the cluster properly.

```
## run a test on the local platform
run(x='sleep_pipe', platform='local', execute=FALSE)
## run a test on the HPCC platform (torque, sge, moab, slurm, lsf)
run(x='sleep_pipe', platform='torque', execute=TRUE)
```

NOTE: In case the test is not successful, please follow the advanced configuration page for more details on how to solve the issues.

2 Fetch and download the pipeline

Next, we will download a pipeline which processes multiple fastq files of a sample into a single aligned and merged BAM file.

```
cd ~/flowr/pipelines
base=https://raw.githubusercontent.com/sahilseth/flowr/devel/inst/pipelines
wget $base/fastq_bam_bwa.R
wget $base/fastq_bam_bwa.conf
wget $base/fastq_bam_bwa.def
```

2.1 Download data/genome

2.1.1 Reference Genome

One can download the reference genome including indexes of various alignment tools from Illumina's iGenomes website.

You may skip this step, if you already have the genome fasta and related files.

```
mkdir ~/flowr/genomes; cd ~/flowr/genomes
url=ussd-ftp.illumina.com/Homo_sapiens/NCBI/build37.2/Homo_sapiens_NCBI_build37.2.tar.gz
ftp ftp://igenome:G3nom3s4u@$url
tar -zxvf Homo_sapiens_NCBI_build37.2.tar.gz
```

A typical NGS pipeline consists of many steps, each with several parameters. You can modify `fastq_bam_bwa.conf`, specifying paths to various tools and their default options (samtools, bwa, picard and reference genome indexes).

Note: All parameters of this pipeline are conveniently specified in a tab-delimited format in the **fastq_bam_bwa.conf** file.

```
## customize parameters, including paths to samtools, bwa, reference genomes etc.
vi fastq_bam_bwa.conf
```

2.1.2 Example data

You may skip this step if you already have raw reads for a sample, in fastq format.

```
mkdir ~/flowr/genomes; cd ~/flowr/genomes
## for testing puposes one may download example fastq files:
wget http://omixon-download.s3.amazonaws.com/target_brca_example.zip
unzip target_brca_example.zip
```

3 Customize flow definition, describing the computing cluster

Next, we need to customize the resource requirements based on the computing platform. You may refer to the flow definition format for more details.

```
## customize the resource requirements in flowdef:
- need to change: queue, platform
- may change: walltime, memory, CPUs etc.
vi fastq_bam_bwa.def

## read check flowdef (shell)
flowr as.flowdef x=fastq_bam_bwa.def

## OR from R
as.flowdef(x='fastq_bam_bwa.def')
```

Read and check flowdef

jobname	sub_type	prev_jobs	dep_type	queue	memory_reserved	walltime	cpu_reserved	platform	jobid
aln1	scatter	none	none	medium	16384	2:00	12	lsf	1
aln2	scatter	none	none	medium	16384	2:00	12	lsf	2
sampe	scatter	aln1,aln2	serial	medium	16384	2:00	1	lsf	3
fixrg	scatter	sampe	serial	medium	16384	2:00	1	lsf	4
merge	serial	fixrg	gather	medium	16384	12:00	1	lsf	5

A flow definition with default values has already been supplied, briefly,

- Submission Type (**sub_type**): determines, how each step is **submitted** to the cluster. All steps except `merging` may have multiple subprocess (each of which can run in parallel). Thus, we spread (`scatter`) them across the cluster.
- Previous Jobs (**prev_jobs**): The two `aln` steps of `bwa` may be run in parallel, and its subsequent `sampe` would wait for both. Spcecifically, in case of multiple fastq files i^{th} `sampe` step would wait for i^{th} `aln1` and `aln2` steps.
- **cpu_reserved**: Since `aln` can use multiple cores, we provide it 12 cores, and for rest of the steps, 1 core each.
- **walltime**: Merging may take a little longer, so we give it ample amount of time (12 hours). Some computing platforms specify time as `hh:mm:ss` and others prefer `hh:mm`, you may need to check with your system admin.
- **memory**: For simplicity we can assign 16GB (16000kb) of memory to each of these steps (may be an overkill, please change as necessary).
- **queue**: We use a generic `medium` queue, since if usually exists; please change as needed.
- **platform**: Finally, specify the platform of your computing cluster (moab, lsf, torque, sge, slurm [alpha])

Tip: Once we define the flow definition correctly, we may not need to change it any further (one time effort).

4 Submit to cluster

4.1 Single step cluster submission

Note: Assuming that the pipeline along with its `.def` and `.conf` files is available in `~/flowr/pipelines`. Also, `.conf` files should have all the correct paths and `.def` file should have resource requirements specified correctly.

```
## get input fastqs
fqs1=~/flowr/genomes/target_brca_example/brca.example.illumina.0.1.fastq
fqs2=~/flowr/genomes/target_brca_example/brca.example.illumina.0.2.fastq

## submit to the cluster
flowr run x=fastq_bam_bwa fqs1=$fqs1 fqs2=$fqs2 samplename=samp execute=TRUE

## change the platform specified in flowdef
flowr run x=fastq_bam_bwa fqs1=$fqs1 fqs2=$fqs2 samplename=samp execute=TRUE platform=slurm
```

OR from R using:

```
library(flowr)
fqpath = "~/flowr/genomes/target_brca_example"
## demonstrating that multiple fqs can be used here...
fobj = run(x = "fastq_bam_bwa", samplename = "samp1", execute = TRUE,
        fqs1 = rep(paste0(fqpath, "/brca.example.illumina.0.1.fastq"), 2),
        fqs2 = rep(paste0(fqpath, "/brca.example.illumina.0.2.fastq"), 2))
```

Refer to the help pages for more details on the run function.

4.2 (Optional) Details regarding cluster submission

The **run** function performs several steps, finally submitting the commands to the cluster. It may be useful to go through these steps to understand the details.

1. Get user inputs

Using the name of the pipeline, `run` fetches it in various places inclusing `~/flowr/pipelines`.

```
library(flowr)
setwd("~/flowr/pipelines")
source("fastq_bam_bwa.R")
## this may throw a warning if paths do not exist
## if you have used modules instead of full paths please ignore the warnings
load_opts("fastq_bam_bwa.conf")

## Get example input
## these can be a vector of multiple paired-end files
## OR multiple single-end files
fqs1 = "~/flowr/genomes/target_brca_example/brca.example.illumina.0.1.fastq"
fqs2 = "~/flowr/genomes/target_brca_example/brca.example.illumina.0.2.fastq"
samp = "samplename"
```

```
## optionally specify the center, lane, platform etc.
set_opts(rg_center = "the_institute", rg_lane = "1")

## **Note:** load_opts checks if variables ending with
## _exe, _path, _dir etc. exist or not.
## make sure they are all correct.
## Ignore the warnings, if instead of specifying full path to a tool
## you are using the module command.
```

Refer to the help pages of fetch_pipes and fetch_pipes for more details.

2. Read flow definition

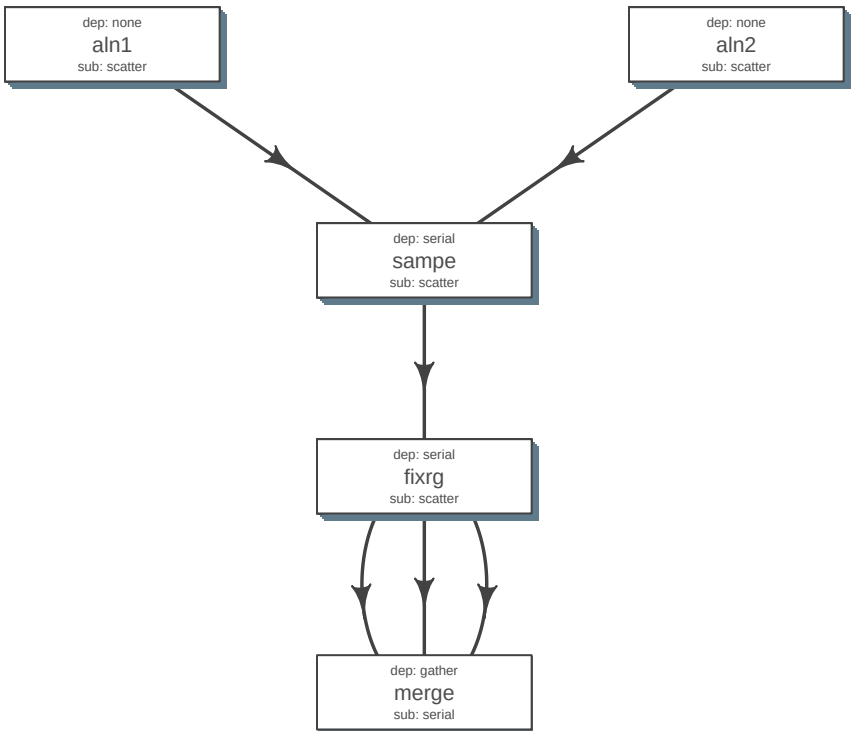
```
def = as.flowdef("fastq_bam_bwa.def")
```

```
## def seems to be a file, reading it...
## --> checking if required columns are present...
## --> checking if resources columns are present...
## --> checking if dependency column has valid names...
## --> checking if submission column has valid names...
## --> checking for missing rows in def...
## --> checking for extra rows in def...
## --> checking submission and dependency types...
```

The plot would work only if you have X11 etc enabled, i.e. if you logged into the cluster using `ssh -X` (or `ssh -Y`).

Optionally, one can edit all config files on their own machine, debug and sort issues; when done, upload them to the cluster and submit.

```
plot_flow(def) ## on a cluster, only works if graphics X11 is enabled. ssh -X
```



3. Create a table with all commands to run

We use the function `fastq_bam_bwa` to create a flow mat.

```
## run the module and create a flow mat, with all the commands
out = fastq_bam_bwa(fqs1, fqs2, samplename = samp)

## optionally, write this to a file (a simple tab delimited table)
write_sheet(out$flowmat, "fastq_bam_bwa.tsv")
```

4. Executing on the computing cluster

Now we can submit this to the cluster using:

```
fobj2 = to_flow(x='~/flowr/pipelines/fastq_bam_bwa.tsv',
                def='~/flowr/pipelines/fastq_bam_bwa.def',
                name = "fastq_bam_bwa",
                execute=TRUE)
```

OR from the terminal using:

```
flowmat=~/flowr/pipelines/fastq_bam_bwa.tsv
flowdef=~/flowr/pipelines/fastq_bam_bwa.def
flowr to_flow x=$flowmat def=$flowdef name=fastq_bam_bwa execute=TRUE
```

Tip: This example shows a single sample, but you may have as many samples in the flowmat. In case of multiple samples, the **samplename** column is used to group commands and each set if submitted as a individual flow.

Several other functions, one may use after submission:

checking the status:

```
## from R:
status(x="~/flowr/runs/fastq_bam_bwa*")

## OR from terminal using:
flowr status x=~/flowr/runs/fastq_bam_bwa*
```

```
|          | total| started| completed| exit_status|status     |
|:---------|-----:|-------:|---------:|-----------:|:----------|
|001.aln1  |     1|       1|         0|           0|processing |
|002.aln2  |     1|       1|         0|           0|processing |
|003.sampe |     1|       0|         0|           0|pending    |
|004.fixrg |     1|       0|         0|           0|pending    |
|005.merge |     1|       0|         0|           0|pending    |
```

Additionally, you may **kill** or **rerun** the flow as well.

```
flowr kill x=~/flowr/runs/fastq_bam_bwa*
flowr rerun x=~/flowr/runs/<full path of the flow> start_from=fixrg
```

Please use the respective help pages for more details.