Variant calling from RNA-seq data using the GATK joint genotyping workflow

Jean-Simon Brouard

April 16, 2021

Abstract

Put the abstract here.

1 Introduction

Since the introduction of RNAseq, many researchers have seen the opportunity to use this data not only for differential expression analysis, but also for calling variants Piskol et al. [2013]. Examples of such researches include a, b, c and d. Whereas trusted bioinformatic protocols exist for detecting sequence variants on a variety of DNAseq samples (germline DNA, whole-exome sequencing etc.) that come from distinct contexts Koboldt [2020], environnemental samples), protocols designed to handle RNAseq data are scarce Piskol et al. [2013]. At present the gold-standard for variant calling on RNAseq data is the GATK Per-sample workflow although an updated detailed workflow documentation for calling variants in RNAseq data is in the roadmap of the GATK experts GATK [2021a].
Currently, researchers interested in performing GATK variant calling on RNAseq data have the option of using the fully validated Per-sample workflow GATK [2021b] or using an in-progress advanced workflow designed for cloud computing GATK [2021c]. As mentionned, the Per-sample approach has several drawbacks (cite myself)
An appealing alternative would be to follow most of the GATK Best practices relative to RNAseq data and to take advantage of joint genotyping approach which is avalaible in version 3 and 4 of GATK for germline short variants and indels (ref). The joint-genotyping method has proven to be more sensitive, more flexible and to reduce computational challenges relative to the traditional calling approach GATK [2021d]. In addition, the latter approach has the advantage to facilitate the inceremental discovery of variants that origin from distinct cohorts of samples. Technically, this can be achieved by combining parts of the GATK RNAseq workflow and parts of the GATK joint genotyping workflow.
In spite that the protocol described here largely use workflows and concepts developped by the GATK team, the reader should be aware that it has not been

validated by GATK experts. We have shown previously that a similar workflow has and we did not ahve experienced any problesm (Brouard)
Here we present GATK4 and fully updated version of this approach.
an update end-to-end analysis of...
Figure 1 present the workflow proposed here.
In the next section, we describe how the diverse programs required to perform the whole analysis can be installed.

2 Materials

2.1 Environment

The vast majority of commands in this tutorial have been carefully tested and fully executed on a remote linux server working with the Sun Grid Engine (SGE) workload manager. Obviously, batch scripts will need to be slightly adapted if another workload manager is in place on your computer cluster or if you intend to perform the analysis locally on a linux machine.

2.2 Installing bioinformatic programs

2.2.1 The GATK suite

It is suggested to install the gatk4 suite in a separate conda environment. Assuming that you are familiar with the conda package management system, you could install all GATK programs in a environment called 'gatk4' with the following command:

```
conda create -n gatk4 -c bioconda gatk4
```

2.2.2 The SRA toolkit programs

You can easily download public sequences from the NCBI Sequence Read Archive (SRA) using the NCBI SRA toolkit. Detailed instructions about this tool can be found at https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=toolkit_doc.. However, before using it, do not forget to configure the SRA toolkit program (https://github.com/ncbi/sra-tools/wiki/03.-Quick-Toolkit-Configuration.
Once installed, export the SRA toolkit programs in you PATH:

```
export PATH=$PATH:$PWD/sratoolkit.2.10.9-ubuntu64/bin
```

You can make this change persistent, by adding the previous line to your .bashrc file.

2.2.3 The STAR aligner

The best practices from GATK recommand to align RNA-seq reads with STAR (ref).
Although you can retrieve and install the STAR aligner with conda, it can be installed easily by just downloading the latest STAR source from releases:

```
wget https://github.com/alexdobin/STAR/archive/2.7.7a.zip
unzip 2.7.7a.zip
cd STAR-2.7.7a/
```

You can then safely use the pre-compiled STAR executables located in the bin/ subdirectory. It is convenient to add the executables to your PATH:

```
export PATH=$PATH:$PWD/bin/Linux_x86_64
```

2.2.4 The Picard tools

We will use the Picard tools to mark duplicated reads. One could find more information about the picard tools at: https://broadinstitute.github.io/picard/.
One can download the pre-build java program with:

```
wget https://github.com/broadinstitute/picard/releases/download/2.25.0/picard.jar
```

It is recommend to set up an environment variable to act as a shortcut. To make it persistent, simply, add a line to your .bashrc file:

```
export PICARD=/home/AAFC-AAC/brouardjs/bioinfo_programs/picard.jar
```

Then, you would be able to call Picard tools with:

```
java -jar $PICARD
```

2.2.5 Samtools, BCFtools and HTSlib

The Samtools, the BCFtools and the HTSlib are now divided in three separated repositories and can be found at http://www.htslib.org/http://www.htslib.org/.
Mention the 2 2021 references that can be found on the htslib/documentation.
The Samtools are the gold-standard programs to read, write, edit, index and view alignments files in the SAM, BAM and CRAM format. In the same way, the BCFtools are the best option to manipulate sequence variants stored in the BCF2, VCF and gVCF format. The HTSlib are a C library designed to read and write high-throughput sequencing data that is used by both the Samtools and the BCFtools. Additionnaly, the HTSlib contains the tabix and bgzip utilities that are mandatory to work with vcf files.
The most straightforward way to make use of these three distinct packages is to build them independently.
Use the commands below to install the Samtools:

```
wget https://github.com/samtools/samtools/releases/download/1.11/samtools-1.11.tar.bz2
bzip2 -d samtools-1.11.tar.bz2
tar -xvf samtools-1.11.tar
cd  samtools-1.11
./configure --prefix=$HOME/bioinfo_programs/bcftools-1.12
```

And you may wish to add the
bin directory to your $PATH:

```
export PATH=$PATH=/home/AAFC-AAC/brouardjs/bioinfo_programs/bcftools-1.12/bin
```

The BCFtools and HTSlib can be installed with analogous commands.

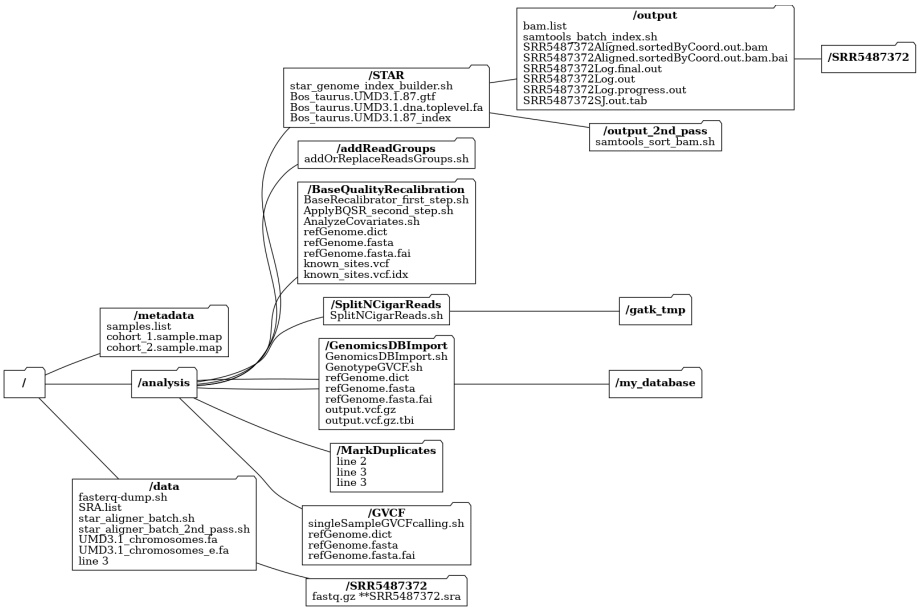

Figure 1: long desc.

3 Methods

3.1 Downloading scripts used in this tutorial

If one is interested to reproduce the workflow presented here, a good starting point would be to download the following GitHub repository:

```
git clone https://github.com/soda460/RNAseq_GATK_JGW.git
```

It contains all scripts described in this section as well as several text files that allow to easily reproduce the analysis presented here. Note that some scripts use relative paths. Therefore using the files/folder organization presentend in figure 2 will help to reproduce the results presented here with minimal changes. contained will also make the whole process easier.

3.2 Downloading the MAP dataset from Sequence Read Archive

We will use the complete RNA-seq raw sequences from [Ariel et al., 2021]. However, rather than using the 72 samples presented in the article, we will focus on 24 representative samples of this study:
The **SRA.list** file contains the SRA identifiers of these 24 samples:

```
SRS2153774
SRS2153779
SRS2153781
SRS2153786
SRS2153787
SRS2153791
SRS2153793
SRS2153797
SRS2153799
SRS2153803
SRS2153805
SRS2153809
SRS2153811
SRS2153815
SRS2153817
SRS2153821
SRS2153823
SRS2153827
SRS2153829
SRS2153833
SRS2153835
SRS2153839
SRS2153841
SRS2153845
```

Navigate to the /data directory and download the 24 samples with the the following command from the SRAtoolkit:

```
prefetch --option-file SRA.list
```

To extract fastq files from .sra files, the fasterq-dump command is required. For example, the following command will produce SRR5487396_R1.fastq.gz and SRR5487396_R2.fastq.gz files from the SRR5487396.sra archive.

```
fasterq-dump --split-files SRR5487396/SRR5487396.sra
```

In practice, you will want to extract all downloaded sequence read archives, which are nested in distinct folders. Navigate to the /data folder and use the following qsub command to lauch the faster-qdump commands sequentially:

```
qsub faster-qdump.sh
```

where **faster-qdump.sh** is a bash script containing instructions for the SGE workload manager and the code to iterate on all downloaded sequence read archives and produce the forward and reverse fastq files:

```
#! /bin/bash
#$ -S /bin/bash
```

```
#$ -cwd
#$ -N 'fasterq-dump'
#$ -pe smp 12
#$ -o ./qsub_log.txt
#$ -e ./qsub_err.txt

for i in `ls -d SRR*`; do
cd $i
fasterq-dump --split-files $i.sra
cd ..
done
```

However, execute the latter script would take a lot of times. To take advantage of the capacity of the cluster, one should may consider launching array tasks (task in parallel). To learn more about this, one can visit this page (SGE). Before using the faster-qdump_parallel.sh, we need to create a simple list file of the 24 SRR* folders present in the /data folder:

```
ls -d1 SRR5487*>SRR.list
```

Be sure that the SRR.list file contains the 24 SRR identifiers (without the .sra extension) on separates lines before lauching the paralle version of fasterq.dump.sh which look like:

```
#!/bin/bash
#$ -V
#$ -N fasterq-dump
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e ./qsub_err.txt
#$ -o ./qsub_log.txt
#$ -q all.q
#$ -t 1-24
#$ -pe smp 4

input=$(head -n $SGE_TASK_ID SRR.list | tail -n 1)

fasterq-dump --split-files $input/$input.sra
```

Figure x show how a computationnaly intense task, namely the decompressing of an .sra archive, can be run at the same time with different input files. This can be done by using the SGE task array capabilities. Technically, the same script is run multiple times, with different values taken by the single environment variable $SGE_TASK_ID. Since many bioinformatic tasks used in this tutorial are computationally intensive, most of the scripts presented thereafter will be base on this approach.

3.3 Performing STAR alignment

3.3.1 Generating genome indexes files

STAR genomes are available for a limited number of genomes on the gingeras lab. We will use the for *Bos taurus* UMD3.1.87 annotations file since the BosTau7 version of the genome was used in Ariel et al. [2021]. The GTF file describe all exons whereas the .dna.toplevel.fa fasta file contains the corresponding sequences.

```
wget http://labshare.cshl.edu/shares/gingeraslab/www-data/dobin/\
STAR/STARgenomes/Old/ENSEMBL/bos_taurus/Bos_taurus.UMD3.1.87.gtf;

wget http://labshare.cshl.edu/shares/gingeraslab/www-data/dobin/\
STAR/STARgenomes/Old/ENSEMBL/bos_taurus/Bos_taurus.UMD3.1.dna.toplevel.fa;
```

To prepare genome index files for STAR, use the genomeGenerate built-in STAR command as in the following script:
star_genome_index_builder.sh

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N 'STAR_genome_builder'
#$ -pe smp 6
#$ -o ./qsub_log.txt
#$ -e ./qsub_err.txt
STAR --runThreadN 6 \
--runMode genomeGenerate \
--genomeDir Bos_taurus.UMD3.1.87_index \
--genomeFastaFiles ./Bos_taurus.UMD3.1.dna.toplevel.fa \
--sjdbGTFfile ./Bos_taurus.UMD3.1.87.gtf \
--sjdbOverhang 99
```

3.3.2 Aligment with the STAR aligner

At this step, we align our raw .fastq files with the STAR aligner.

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N 'STAR_aligner'
#$ -pe smp 12
#$ -o ./qsub_log.txt
#$ -e ./qsub_err.txt

for i in `ls -d SRR*`; do
echo "Performing STAR alignemnt with $i"
```

```
mkdir -p ../analysis/STAR/output/$i
STAR --genomeDir ../analysis/STAR/Bos_taurus.UMD3.1.87_index \
--runThreadN 12 \
--readFilesIn  ./$i/$i"_1.fastq" ./$i/$i"_2.fastq" \
--outFileNamePrefix ../analysis/STAR/output/$i \
--outSAMtype BAM SortedByCoordinate \
--outSAMunmapped Within \
--outSAMattributes Standard
```

3.3.3 Adding Splice junctions (2nd mapping pass)

GATK workflow recommend to perform 2-pass mapping with the STAR aligner. The 2nd mapping pass is identical to the first alignment except that all splice junctions discovered in the first pass are given to the programs. Simply specify a list of all splice junctions files after the –sjdbFileChrStartEnd argument.

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N 'STAR_aligner_2nd_pass'
#$ -pe smp 12
#$ -o ./qsub_log.txt
#$ -e ./qsub_err.txt

for i in `ls -d SRR*`; do
echo "Performing STAR alignemnt (2nd pass) with $i"
mkdir -p ../analysis/STAR/output_2nd_pass/$i
STAR --genomeDir ../analysis/STAR/Bos_taurus.UMD3.1.87_index \
--runThreadN 12 \
--readFilesIn  ./$i/$i"_1.fastq" ./$i/$i"_2.fastq" \
--outFileNamePrefix ../analysis/STAR/output_2nd_pass/$i \
--outSAMtype BAM SortedByCoordinate \
--outSAMunmapped Within \
--outSAMattributes Standard \
--sjdbFileChrStartEnd \
../analysis/STAR/output/SRR5487372SJ.out.tab \
../analysis/STAR/output/SRR5487384SJ.out.tab \
../analysis/STAR/output/SRR5487396SJ.out.tab \
../analysis/STAR/output/SRR5487408SJ.out.tab \
../analysis/STAR/output/SRR5487420SJ.out.tab \
../analysis/STAR/output/SRR5487432SJ.out.tab \
../analysis/STAR/output/SRR5487376SJ.out.tab \
../analysis/STAR/output/SRR5487388SJ.out.tab \
../analysis/STAR/output/SRR5487400SJ.out.tab \
../analysis/STAR/output/SRR5487412SJ.out.tab \
../analysis/STAR/output/SRR5487424SJ.out.tab \
```

```
../analysis/STAR/output/SRR5487436SJ.out.tab \
../analysis/STAR/output/SRR5487378SJ.out.tab \
../analysis/STAR/output/SRR5487390SJ.out.tab \
../analysis/STAR/output/SRR5487402SJ.out.tab \
../analysis/STAR/output/SRR5487414SJ.out.tab \
../analysis/STAR/output/SRR5487426SJ.out.tab \
../analysis/STAR/output/SRR5487438SJ.out.tab \
../analysis/STAR/output/SRR5487382SJ.out.tab \
../analysis/STAR/output/SRR5487394SJ.out.tab \
../analysis/STAR/output/SRR5487406SJ.out.tab \
../analysis/STAR/output/SRR5487418SJ.out.tab \
../analysis/STAR/output/SRR5487430SJ.out.tab \
../analysis/STAR/output/SRR5487442SJ.out.tab
done
```

3.3.4 Sorting and indexing alignment files

After you align your sequences. You will want that your .bam files to be indexed and sorted by coordinates. As mentioned earlier, STAR allows the output to be sorted by coordinates. If you have carefully followed this tutorial, there is no need to sort again your bam files and you can safely ignore the next step. However, if for any reasons, your aligmnents files are not sorted adequately, the sort option of samtools is as easy as :

```
samtools sort file.bam -o file_sorted.bam
```

For our data, a single qsub command with the following script will sort the operations sequentially:

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
for i in `ls SRR5487400Aligned*.bam | xargs basename -s '.bam'`; do
 samtools sort $i.bam -o $i"_sorted.bam"
done
```

In the STAR output folder, prepare a list of bam files to be indexed with:

```
ls -1 *.bam > bam.list
```

then, you can index all alignements in parallel with:

```
qsub samtools_batch_index.sh
```

where the latter script looks like:

```
#!/bin/bash
#$ -V
#$ -N samtools_index
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o o
#$ -q all.q
#$ -t 1-24
#$ -pe smp 2

input=$(head -n $SGE_TASK_ID samples.list | tail -n 1)

samtools index $input
```

3.4 Add Read groups

Before running any GATK workflow, one will need to add Read groups. This step ensure that relevant informations about the sequencing process are assigned to each read in an alignment file. When this step is done carefully, it allow to correct sequencing biais that could arise from...
Here, we will set the flowcells, sequencing lanes and sample barcode in the platform unit (PU) tag.
Before running the main script, we will extract some important columns from our metadata file and place them in separate files that will be used to populate important fields of the read groups. Here our metadata .csv file:

```
sample,cowID,SRAID,Run,RGPU
A_CTL24,cowA,SRS2153774,SRR5487372,C5NL3ACXX.1.CAGATC
A_MAP24,cowA,SRS2153779,SRR5487376,C5NL3ACXX.1.TGACCA
B_CTL24,cowB,SRS2153781,SRR5487378,C5NL3ACXX.3.TGACCA
B_MAP24,cowB,SRS2153786,SRR5487382,C5NL3ACXX.3.GTGAAA
C_CTL24,cowC,SRS2153787,SRR5487384,C5K8FACXX.3.AGTCAA
C_MAP24,cowC,SRS2153791,SRR5487388,C5K8FACXX.3.GTCCGC
D_CTL24,cowD,SRS2153793,SRR5487390,C5K8FACXX.1.TGACCA
D_MAP24,cowD,SRS2153797,SRR5487394,C5K8FACXX.1.GTGAAA
E_CTL24,cowE,SRS2153799,SRR5487396,C547FACXX.1.AGTCAA
E_MAP24,cowE,SRS2153803,SRR5487400,C547FACXX.1.GTCCGC
F_CTL24,cowF,SRS2153805,SRR5487402,C547FACXX.3.TGACCA
F_MAP24,cowF,SRS2153809,SRR5487406,C547FACXX.3.GTGAAA
G_CTL24,cowG,SRS2153811,SRR5487408,C5NL3ACXX.5.AGTCAA
G_MAP24,cowG,SRS2153815,SRR5487412,C5NL3ACXX.5.GTCCGC
H_CTL24,cowH,SRS2153817,SRR5487414,C5NL3ACXX.7.TGACCA
H_MAP24,cowH,SRS2153821,SRR5487418,C5NL3ACXX.7.CTTGTA
```

```
I_CTL24,cowI,SRS2153823,SRR5487420,C5K8FACXX.5.AGTTCC
I_MAP24,cowI,SRS2153827,SRR5487424,C5K8FACXX.5.GTGAAA
J_CTL24,cowJ,SRS2153829,SRR5487426,C5K8FACXX.7.TGACCA
J_MAP24,cowJ,SRS2153833,SRR5487430,C5K8FACXX.7.GTGAAA
K_CTL24,cowK,SRS2153835,SRR5487432,C547FACXX.5.AGTCAA
K_MAP24,cowK,SRS2153839,SRR5487436,C547FACXX.5.GTCCGC
L_CTL24,cowL,SRS2153841,SRR5487438,C547FACXX.7.AGTCAA
L_MAP24,cowL,SRS2153845,SRR5487442,C547FACXX.7.GTCCGC
```

The RGLB.txt file will hold the library name. In this case, we will use the sample name

```
# Corresponding to the sample name
cut -f 1 -d ',' ../../metadata/metadata.txt | tail -n +2 > RGLB.txt

# Corresponding to the PU tag FLOW CELL
cut -f 5 -d ',' ../../metadata/metadata.txt | tail -n +2 > RGPU.txt
```

```
#!/bin/bash
#$ -V
#$ -N AddOrReplaceReadGroups
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o logfile.txt
#$ -q all.q
#$ -t 1-24
#$ -pe smp 1

eval "$(conda shell.bash hook)"
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/STAR/output_2nd_pass"
OUTPUT="$HOME/jsb/springer/analysis/addReadGroups"

readarray -t RGLB < ./RGLB.txt
readarray -t RGPU < ./RGPU.txt

input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

java -jar $PICARD AddOrReplaceReadGroups \
I=$BAMPATH/$input"Aligned.sortedByCoord.out.bam" \
O=$OUTPUT/$input".bam" \
RGLB=$RGLB \
```

```
RGPL=ILLUMINA \
RGPU=$RGPU \
RGSM=$input

conda deactivate
```

To validate that the reads groups have been correctly, you could use this simple trick on the .bam files:

```
samtools view -H sample.bam | grep '@RG'
```

3.5 MarkDuplicates

Duplicate reads can arise from PCR duplication artifacts that take place during the library construction or from reading errors that occur during the sequencing process (optical duplicates). Regardless of their origin, these reads need to be identified in alignment files. The MarkDuplicate program from the picard tools have many options to deal with these issue and output some metrics. For example, the program offer the possibility to completey remove the duplicate reads and to make the distinction between optical and PCR duplicates.
Since this tool require a
Here we simply identified the duplicate reads:

```
#!/bin/bash
#$ -V
#$ -N samtools_index
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -o logfile.$TASK_ID.log
#$ -q all.q
#$ -t 1-24
#$ -pe smp 2

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/addReadGroups"
OUTPUT="$HOME/jsb/springer/analysis/MarkDuplicates"

input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

java -jar $PICARD MarkDuplicates \
I=$BAMPATH/$input".bam" \
O=$OUTPUT/$input"_marked_duplicates.bam" \
M=$OUTPUT/$input"_marked_dup_metrics.txt"
```

BaseRecalibration IndexFeatureFile
gatk IndexFeatureFile -input jsb.vcf

3.5.1 The gff3 file

With the command below, one can see the number of features annotated for each ID in the gff3 file:

```
{bash}
cat Ensembl79_UMD3.1_genes.gff3 | cut -f 1 | uniq -cd
```

The result of this command is:

```
21287 GK000001.2
18819 GK000030.2
26235 GK000002.2
29793 GK000003.2
29598 GK000005.2
18201 GK000004.2
15375 GK000006.2
16534 GK000008.2
28038 GK000007.2
24860 GK000011.2
12861 GK000009.2
23581 GK000010.2
9839 GK000012.2
17254 GK000015.2
11207 GK000014.2
18771 GK000013.2
17134 GK000016.2
14916 GK000017.2
8554 GK000020.2
13262 GK000021.2
26279 GK000018.2
30597 GK000019.2
7463 GK000024.2
15631 GK000022.2
14696 GK000023.2
10159 GK000026.2
14729 GK000029.2
8317 GK000028.2
5227 GK000027.2
17365 GK000025.2
14 GJ058430.1
31 GJ058425.1
3 GJ059486.1
4 GJ058433.1
16 GJ058437.1
15 GJ058424.1
12 GJ060129.1
```

```
12 GJ059670.1
124 AY526085.1
24 GJ059556.1
9 GJ059463.1
13 GJ060118.1
19 GJ060120.1
16 GJ060032.1
23 GJ059509.1
3 GJ058729.1
3 GJ060027.1
3 GJ058256.1
```

Therefore, to ensure that the gff3 chromosome IDs match those presennt in our alignment files, we need to slightly edit this file to replace the pattern used for the chromosome IDs (GK + 0000 + chromosome number) by solely the chromosome number. This could be done easily with sed:

```
sed -r s'/^GK[0]+([0-9]+).2/\1/'g Ensembl79_UMD3.1_genes.gff3 > Ensembl79_UMD3.1_genes_e.gf
```

Obvisouly it is good idea to inspect the edited file to ensure that the desired changes have been done properly.

3.6 SplitNCigarReads

The next step is very specific to RNAseq. Briefly, during this step reads that match align over distinct exons are split and blah.
However, before doing this step, we need to get and prepare reference genome files.

3.6.1 Prepare reference genome files

Until now, we have work with STAR reference genome which barely contains sequences that are transcribed in RNA. In order to produce alignments files that will adequately represent reads that span splice junctions, the SplitNCigarReads command require the complete reference sequence. At this point, pay attention to download the same version of the refence genome as the one that was used to prepare the STAR genome.
To get the UMD3.1 assembly, visit https://bovinegenome.elsiklab.missouri.edu/ and download the UMD3.1_chromosomes.fa.gz and Ensembl79_UMD3.1_genes.gff3.gz files. If you inspect the headers of the UMD3.1_chromosomes.fa, you will notice that each fasta entry contains many fields (e.g. gnl , UMD3.1 Accession numbers). To inspect the fasta headers, simply use grep:

```
grep ">" UMD3.1_chromosomes.fa
```

```
>gnl|UMD3.1|GK000010.2 Chromosome 10 AC_000167.1
>gnl|UMD3.1|GK000011.2 Chromosome 11 AC_000168.1
```

```
>gnl|UMD3.1|GK000012.2 Chromosome 12 AC_000169.1
...
>gnl|UMD3.1|GJ060407.1 GPS_000344847.1 NW_003101152.1
>gnl|UMD3.1|GJ060408.1 GPS_000344848.1 NW_003101153.1
```

It is crucial that entries in the reference genome match the corresponding ones in the STAR genome. One could use the following commands to rename the complicated chromosome entries from the UMD3.1 genome to the plain chromosome numbers as what is seen in the STAR genome:

```
sed -r s'/^>.+Chromosome\s+(\S+)\s+.+/>\1/' UMD3.1_chromosomes.fa > temp1.fa
grep ">" temp1.fa | head -n 40
sed -r s'/^>gnl\|UMD3\.1\|(\S+)+\s+.+/>\1/' temp1.fa > temp2.fa
grep ">" temp2.fa | tail -n +30 | head -n 10
mv temp2.fa refGenome.fasta
rm temp1.fa
```

Along with the reference genome, many GATK tools will need a dictionary file ending in .dict and an index file ending in .fai. https://gatk.broadinstitute.org/hc/en-us/articles/360035531652-FASTA-Reference-genome-format). These files need to share the same basename as the reference genome. You can prepare the index with:

```
samtools faidx refGenome.fasta
```

Use the GATK CreateSequenceDictionary tool to create the required .dict file:

```
gatk CreateSequenceDictionary -R refGenome.fasta
```

3.6.2 Run SplitNCigarReads

Here the listing for the SplitNCigarReads step:

```
#!/bin/bash
#$ -V
#$ -N SplitNCigarReads
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o SplitNCigarReads.$TASK_ID.log
#$ -q all.q
#$ -t 1-24
#$ -pe smp 4

eval "$(conda shell.bash hook)"
```

```
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/MarkDuplicates"
OUTPUT="$HOME/jsb/springer/analysis/SplitNCigarReads"

input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

gatk SplitNCigarReads \
-R refGenome.fasta \
-I $BAMPATH/$input"_marked_duplicates.bam" \
-O $OUTPUT/$input"_SplitNCigarReads.bam" \
--tmp-dir $output/gatk_tmp

conda deactivate
```

3.7 Base Quality Recalibration

This optional step is highly recommended by

3.7.1 BaseRecalibrator first step

```
#!/bin/bash
#$ -V
#$ -N BaseRecalibrator
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o BaseQualityRecalibration_$TASK_ID.log
#$ -q all.q
#$ -t 1-24
#$ -pe smp 4

eval "$(conda shell.bash hook)"
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/SplitNCigarReads"
OUTPUT="$HOME/jsb/springer/analysis/BaseQualityRecalibration"

input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

gatk BaseRecalibrator \
-R ./refGenome.fasta \
```

```
-I $BAMPATH/$input"_SplitNCigarReads.bam" \
--known-sites ./jsb.vcf \
-O $OUTPUT/$input"_recal_data.table"

conda deactivate
```

In the second step of the BaseQuality recalibration, GATK use the content in the .table files to recalibrate base quality.

3.7.2 BaseRecalibrator second step

```
#!/bin/bash

#$ -V
#$ -N ApplyBQSR
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o ApplyBQSR_$TASK_ID.log
#$ -q all.q
#$ -t 1-24
#$ -pe smp 4

eval "$(conda shell.bash hook)"
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/SplitNCigarReads"
OUTPUT="$HOME/jsb/springer/analysis/BaseQualityRecalibration"


echo $SAMPLES
input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

gatk ApplyBQSR \
-R ./refGenome.fasta \
-I $BAMPATH/$input"_SplitNCigarReads.bam" \
--bqsr-recal-file $OUTPUT/$input"_recal_data.table" \
-O $OUTPUT/$input"_recal.bam"

conda deactivate
```

3.8 Variant Calling

More details about the incremental discovery can be found here:

3.8.1 GVCF part

```
#!/bin/bash
#$ -V
#$ -N callingGVCF
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
#$ -e e
#$ -o singleSampleCallingGVCF_$TASK_ID.log
#$ -q all.q
#$ -t 1-24
#$ -pe smp 4

eval "$(conda shell.bash hook)"
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata/samples.list"
BAMPATH="$HOME/jsb/springer/analysis/BaseQualityRecalibration"
OUTPUT="$HOME/jsb/springer/analysis/GVCF"

echo $SAMPLES
input=$(head -n $SGE_TASK_ID $SAMPLES | tail -n 1)

gatk --java-options "-Xmx4g" HaplotypeCaller \
-R ./refGenome.fasta \
-I $BAMPATH/$input"_recal.bam" \
-O $OUTPUT/$input".g.vcf.gz" \
-ERC GVCF

conda deactivate
```

3.8.2 Import DBI

DBI import

```
eval "$(conda shell.bash hook)"
conda activate gatk4

SAMPLES="$HOME/jsb/springer/metadata"
OUTPUT="$HOME/jsb/springer/analysis/GenomicsDBImport"

gatk --java-options "-Xmx4g -Xms4g" \
GenomicsDBImport \
--genomicsdb-workspace-path $OUTPUT/"my_database" \
-L 1 \
```

```
-L 2 \
-L 3 \
-L 4 \
-L 5 \
-L 6 \
-L 7 \
-L 8 \
-L 9 \
-L 10 \
-L 11 \
-L 12 \
-L 13 \
-L 14 \
-L 15 \
-L 16 \
-L 17 \
-L 18 \
-L 19 \
-L 20 \
-L 21 \
-L 22 \
-L 23 \
-L 24 \
-L 25 \
-L 26 \
-L 27 \
-L 28 \
-L 29 \
--sample-name-map $SAMPLES/"cohort_1.sample_map" \
--tmp-dir $HOME/jsb/tmp \
--reader-threads 5
conda deactivate
```

3.8.3 Genotype GVCF

The final step in variant calling can be done with GATK GenotypeGVCFs program.

```
#!/bin/bash
#$ -V
#$ -N GenotypeGVCF
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -b n
```

```
#$ -e e
#$ -o GenotypeGVCF.log
#$ -q all.q
#$ -pe smp 4

eval "$(conda shell.bash hook)"
conda activate gatk4

gatk --java-options "-Xmx4g" GenotypeGVCFs \
-R refGenome.fasta \
-V gendb://my_database \
-O output.vcf.gz
conda deactivate
```

3.9 Examine and visualize alignment files

At this step, you could examine your alignments files.
Tablet (https://ics.hutton.ac.uk/tablet/download-tablet/) is a good program to visualize alignments files. The IVG viewer could also be used.
You should keep a copy of the vcf output file in a safe place. It is fairly common to corrput a .gz file with compressing/decompressing operations.
gunzip -c output.vcf.gz > output.vcf

References

Robert Piskol, Gokul Ramaswami, and Jin Billy Li. Reliable identification of genomic variants from rna-seq data. *American journal of human genetics*, 93: 641–651, October 2013. ISSN 1537-6605. doi: 10.1016/j.ajhg.2013.08.008.

Daniel C. Koboldt. Best practices for variant calling in clinical sequencing. *Genome medicine*, 12:91, Oct 2020.

GATK. Are there best practices for calling variants in rnaseq data?, 2021a. URL `https://gatk.broadinstitute.org/hc/en-us/articles/360035889711-Are-there-Best-Practices-for-calling-variants-in-RNAseq-data-`.

GATK. Rnaseq short variant discovery (snps + indels), 2021b. URL `https://gatk.broadinstitute.org/hc/en-us/articles/360035531192-RNAseq-short-variant-discovery-SNPs-Indels-`.

GATK. gatk4-rnaseq-germline-snps-indels, 2021c. URL `https://github.com/gatk-workflows/gatk4-rnaseq-germline-snps-indels`.

GATK. The logic of joint calling for germline short variants, 2021d. URL `https://gatk.broadinstitute.org/hc/en-us/articles/360035890431-The-logic-of-joint-calling-for-germline-short-variants`.

Olivier Ariel, Jean-Simon Brouard, Andrew Marete, Filippo Miglior, Eveline Ibeagha-Awemu, and Nathalie Bissonnette. Genome-wide association analysis identified both rna-seq and dna variants associated to paratuberculosis in canadian holstein cattle 'in vitro' experimentally infected macrophages. *BMC genomics*, 22:162, Mar 2021.