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

Jean-Simon Brouard, Nathalie Bissonnette

April 29, 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 to quantify gene expression levels, but also for discover genomic variations [1]. 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 [2], protocols designed to handle RNAseq data are scarce [1]. Among all resources available, many rely on the Genome Analysis Toolkit (GATK), an industry standard for variant discovery from next-generation sequencing (NGS) data, developped and maintained by the Broad Institute.

At present the gold-standard for variant calling on RNAseq data is the GATK Per-sample workflow although an updated documentation for calling variants in RNAseq data is in the roadmap of the GATK experts [3]. Currently, researchers interested in performing variant discovery on RNAseq data have the option of using the fully validated Per-sample workflow [4] or using an advanced work-in-progress workflow designed for cloud computing (Google Cloud Platform) [5]. The Per-sample approach has several drawbacks, notably that only variable positions are reported and that the HaplotypeCaller engine cannot leverage population-wide information when calling variants [6, 4].

An appealing alternative would be to use guidelines from the GATK Best practices relative to RNAseq data and to take advantage of joint genotyping approach which is available for the Germline short variant discovery (SNPs + Indels) Best Practices Workflows [7]. The joint-genotyping method has proven to be more sensitive, more flexible and to reduce computational challenges relative to the traditional calling approach [8]. In addition, the latter approach has the advantage to facilitate the incremental 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 (figure 1).

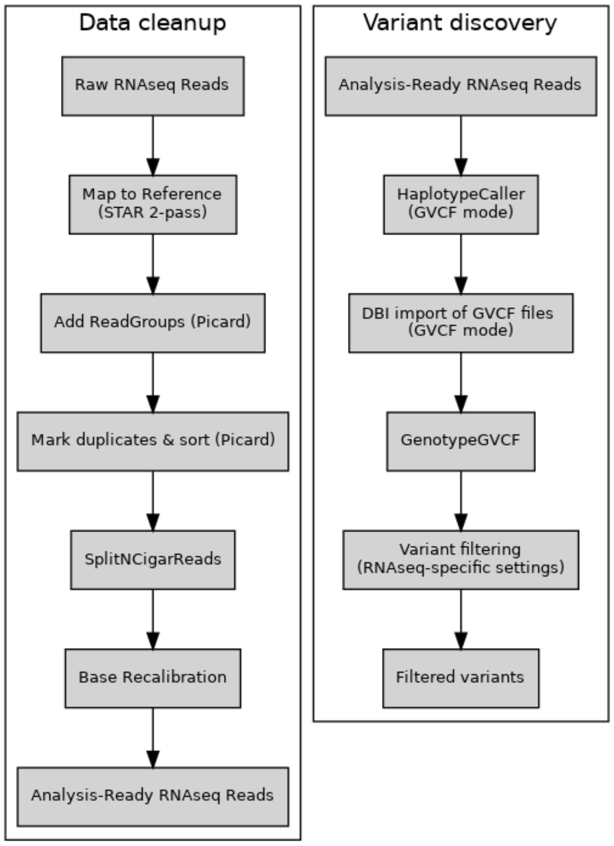

Data cleanup
Raw RNAseq Reads
Map to Reference (STAR 2-pass)
Add ReadGroups (Picard)
Mark duplicates & sort (Picard)
SplitNCigarReads
Base Recalibration
Analysis-Ready RNAseq Reads
Variant discovery
Analysis-Ready RNAseq Reads
HaplotypeCaller (GVCF mode)
DBI import of GVCF files (GVCF mode)
GenotypeGVCF
Variant filtering (RNAseq-specific settings)
Filtered variants

Figure 1: long desc.

In spite that the protocol described here largely use workflows and concepts developped by the GATK team, it should be pointed out that calling variants on RNAseq data with the joint genotyping workflow has still not been validated by GATK experts. We have shown previously that this approach yield similar if not better results when compared to a Per-sample method [6], but one would be advised to carefully and critically examine the results to detect potential artifacts. Here we present a fully updated version of this approach in the form of an end-to-end analysis of RNAseq samples extracted from 50 dairy cows in the context of studying the mechanism of resistance to bovine paratuberculosis.

As shown in figure 1, the workflow presented here involve several steps. In the data cleanup part, the raw RNAseq reads are prepared for RNAseq analysis according to the GATK Best practices relative to RNAseq data. In the variant discovery part, the analyis-ready RNAseq reads follow the germline Joint genotyping workflow up to the variant filtering steps.

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, Bash scripts presented here will need to be slightly adapted to be used with your own raw sequence data. In addition, other small changes will be required to these scripts if another workload manager is in place on your computer cluster or if you intend to perform the analysis on a local machine.

2.2 Installing bioinformatic programs

2.2.1 The GATK suite

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

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

To generate optional plots in the Base Quality Score Recalibration subsection, additionnal libraries are required. Use the following commands to install them alongside GATK in the gatk4 environment:

```
conda activate gatk4
conda install -c conda-forge r-base
conda install -c r r-ggplot2
conda install -c r r-gplots
conda install -c bioconda r-gsalib
```

2.2.2 The NCBI 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
```

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

2.2.3 The STAR aligner

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 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 (https://broadinstitute.github.io/picard/) to mark duplicated reads. 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. Again, to make it persistent in other sessions, simply, add a line to your .bashrc file:

```
export PICARD=$HOME/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 web site (http://www.htslib.org/) contains a plenty of informations about the Samtools suite, a collection of widely used bioinformatic programs designed to read, write, edit, index and view alignments files in the SAM, BAM and CRAM formats [9]. Less known, but just as powerful, the BCFtools are the best option to manipulate sequence variants stored in the BCF2, VCF and gVCF format. Finally, The HTSlib is a C library designed to read and write high-throughput sequencing data that is used by both the Samtools and the BCFtools. The HTSlib notably contains the tabix and bgzip indexing and compression utilities.

Since the Samtools, the BCFtools and the HTSlib projects are now divided in three separated repositories, 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 (and similarly for the BCFtools and HTSlib):

```
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/samtools-1.11
```

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

```
export PATH=$PATH:$HOME/bioinfo_programs/samtools-1.11/bin
```

2.3 Downloading scripts used in this tutorial

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 the next section as well as several text files that allow one to reproduce the analysis presented here. Moreover it contains a structure of files and folders consistent with almost all file paths given in the scripts. As a reference, figure 2 displays the relative organization of the scripts, files and folders refered by, or created by the listings described in this tutorial. Albeit somes files and subfolders were omited, it gives a good overview of the entire workflow.

2.4 Downloading sequences from NCBI SRA

In this tutorial, we choose to work with full-length RNAseq datasets derived from macrophage transcriptomes of cows in response to the infection by *Mycobacterium avium ssp. paratuberculosis* (MAP), the etiological agent of paratuberculosis, or Johne's disease (JD) [10]. A representative subset of 24 samples was selected for which the SRA identifiers can be found in the data/SRA.list file.

```
SRS2153774
SRS2153779
SRS2153781
...
SRS2153841
SRS2153845
```

To download these sequences (221 Gb), navigate to the /data directory and download the samples with the prefetch command from the SRAtoolkit:

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

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

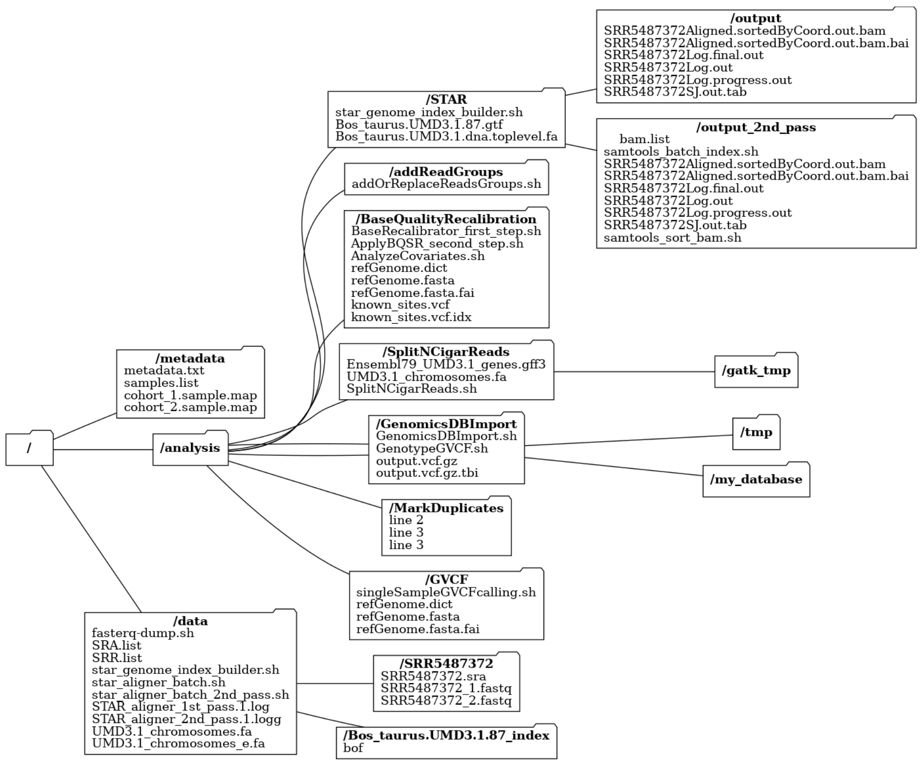


Figure 2: long desc.

```
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 -V -S /bin/bash -cwd -j y -pe smp 12 faster-qdump_sequential.sh
```

where faster-qdump_sequential.sh is a bash script containing instructions for the SGE workload manager and the code to iterate on downloaded archives and to produce the forward and reverse fastq files;

```
#! /bin/bash
#$ -N 'fasterq-dump'
#$ -o ./faster-qdump_log.txt

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

However, executing the latter script would take a lot of times. To take advantage of the parallel computing capacity of compute servers, one may consider launching SGE task array jobs (tasks in parallel). This can be done by using the SGE task array capabilities. Briefly, a single script is run multiple times with different values taken by the single environment variable $SGE_TASK_ID. Using an UNIX trick on a list of samples, each instance of the script can be run on a distinct dataset. Since many bioinformatic tasks used in this tutorial are computationally intensive, most of the scripts presented thereafter will be base on this model. For brevity, we did not include most of the required SGE options (lines beginning with #$ within BASH scripts. Therefore, it is important to include them with the qsub command. For example, in this tutorial, launch parallel tasks (with 24 samples) with:

```
qsub -V -S /bin/bash -cwd -j y -t 1-24 -pe smp 4 parallel_task.sh \
```

Otherwise, omit the -t option and increase the number of cores to be used:

```
qsub -V -S /bin/bash -cwd -j y -pe smp 12 unparallel_task.sh \
```

Before lauching the parallel version of fasterq.dump.sh, create a simple list file of the 24 SRR* folders present in the /data folder:

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

The SRR.list file should contain the 24 SRR identifiers (without the .sra extension) on separates lines.

```
qsub -V -S /bin/bash -cwd -j y -t 1-24 -pe smp 4 faster-qdump_parallel.sh
```

where faster-qdump_parallel.sh reads as follow:

```
#!/bin/bash
#$ -N fasterq-dump
#$ -o ./fasterq-dump.$TASK_ID.log

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

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

3 Methods

3.1 Mapping Reads to Reference (STAR 2-pass)

3.1.1 Generating Genome Indexes Files

The current recommendation of the GATK Per-sample RNAseq workflow is to perform 2-pass mapping with the STAR aligner [11]. STAR genomes are available for a limited number of genomes on the Gingeras lab. Here we will use the *Bos taurus* UMD3.1.87 annotations file since the UMD 3.1.1/bosTau8 version of the genome was used in [10]. The GTF file describes 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. Note that you have to create a directory where STAR could buid the index:

```
mkdir Bos_taurus.UMD3.1.87_index
```

Then, build the index with the star_genome_index_builder.sh:

```
#! /bin/bash
#$ -N 'STAR_genome_builder'
#$ -o ./star_genome_builder_log.txt

STAR --runThreadN 12 \
--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
```

3.1.2 Aligning RNAseq reads (1st mapping pass)

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

```
#!/bin/bash
#$ -N 'STAR_aligner'
#$ -o STAR_aligner_1st_pass.$TASK_ID.log

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

```
mkdir -p ../analysis/STAR/output/$input

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

3.1.3 Aligning RNAseq reads (2nd mapping pass)

The 2nd mapping pass is identical to the first alignment except that all splice junctions discovered in the first pass are given as input to the programs, allowing to detect more reads mapping to novel junctions [11]. Simply specify a list of all splice junctions files after the –sjdbFileChrStartEnd argument.
star_aligner_2nd_pass.sh:

```
#!/bin/bash
#$ -N 'STAR_aligner_2nd_pass'
#$ -o STAR_aligner_2nd_pass.$TASK_ID.log

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

mkdir -p ../analysis/STAR/output_2nd_pass/$input

STAR --genomeDir ./Bos_taurus.UMD3.1.87_index \
--runThreadN 4 \
--readFilesIn  ./$input/$input"_1.fastq" ./$input/$input"_2.fastq" \
--outFileNamePrefix ../analysis/STAR/output_2nd_pass/$input \
--outSAMtype BAM SortedByCoordinate \
--outSAMunmapped Within \
--outSAMattributes Standard \
--outTmpDir ../analysis/STAR/output_2nd_pass/_STARtmp_$SGE_TASK_ID \
--sjdbFileChrStartEnd \
../analysis/STAR/output/SRR5487372SJ.out.tab \
../analysis/STAR/output/SRR5487384SJ.out.tab \
...
../analysis/STAR/output/SRR5487430SJ.out.tab \
../analysis/STAR/output/SRR5487442SJ.out.tab
```

3.1.4 Sorting Alignment Files

After you align your sequences. You will want your BAMfiles 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 and jump to the next subheading. However, if for any reasons, your aligmnents files are not sorted adequately, the sorting by coordinates with the Samtools is as easy as :

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

In a folder containing unsorted alignments, first prepare a list of bam files to be sorted with:

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

For our data, we could re-sort our alignments files with:
samtools_sort_bam.sh:

```
#!/bin/bash
#$ -V
#$ -N samtools_sort
#$ -o samtools_sort.$TASK_ID.log

input=$(head -n $SGE_TASK_ID bam.list | tail -n 1 | xargs basename -s '.bam')

samtools sort $input.bam -o $input"_sorted.bam"
```

Note that this listing could be used as a template to perform a variety of Samtools subcommands that involve renaming output files.

3.1.5 Indexing Alignment Files

BAM files (the binary analog of Sequence Alignment Mapping (SAM) files) are very efficient bioinformatic files designed to store high-throughput alignment data. Typically, they contain the result of the alignment of millions of sequencing reads against a reference genome and greatly benefit from being indexed by genomic positions for faster random access to the aligned reads at a specific locus. In practice most programs will not run in the absence of index files. Obviously, BAM files need to be sorted before being indexed. To index all BAM files present in the STAR output_2nd_pass folder, first prepare a list of files to be indexed with:

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

Then, you can index all alignements in parallel with:
samtools_batch_index.sh:

```
#!/bin/bash
#$ -N samtools_index
#$ -o samtools_index.$TASK_ID.log
```

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

samtools index $input
```

3.2 Adding Read Groups

Contrarily to FASTQ files, SAM files have the capacity to handle large amount of metadata. A good practice would be to always include informations about the reference genome and the samples in aligment files. Informations about the processing steps could also be stored. Many programs, such as GATK, will automatically add this kind of metadata information when manipulating SAM/BAM files.

Most GATK commands involving BAM files require that read groups have been defined. This step ensure that relevant information about the sequencing process follow each read in downstream processing. In addition, when this step is done appropriately, it allow to mitigate the consequences of artifacts associated with sequencing features in the duplicate marking and base recalibration steps [12]. For example, when samples are multiplexed, it is important to known which reads origin from which library, which were sequenced on which flowcell, etc. The GATK engine requires the presence of several read group fields to run without errors. To learn more about the Read groups as understand by the GATK team and to learn how to derive this information from read names, one should consult this page: https://gatk.broadinstitute.org/hc/en-us/articles/360035890671-Read-groups.

Here, we will set the flowcells, sequencing lanes and sample barcodes in the PU (Platform Unit) tag. We will also set the PL (Platform/technology used to produce the read) and LB (DNA preparation library identifier) tags. Note that the ID (Read group identifier) tag is overrided by the PU tag for base recalibration when the latter is defined.

Before running the main script, we will extract the first and fifth columns from our metadata file and place them in separate files, namely RGLB.txt and RGPU.txt, that will be further used to populate the LB and PU read groups fields:

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

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

where /metadata/metadata.txt looks like:

```
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
...
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 therefore hold the library name for each 24 samples (e.g. C_MAP24, which designates a RNAseq library done on the cowC 24h post-infection with the MAP pathogen). The RGPU.txt store the Platform Unit tag, which is made of three types of informations: the flowcell, the sequencing lane and the barcode identifiers, with the following format: FLOWCELL_BARCODE.LANE.SAMPLE_BARCODE.

Use the following listing to add read groups in alignments files. Note that the value given by $SGE_TASK_ID -1 was used to fetch the correct values in RGLB and RGPU arrays because Bash arrays are zero-indexed.
AddOrReplaceReadGroups.sh:

```
#!/bin/bash
#$ -N AddOrReplaceReadGroups
#$ -o logfile.$TASK_ID.log

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

SAMPLES="$HOME/jsb/springer2/RNAseq_GATK_JGW/metadata/samples.list"
BAMPATH="$HOME/jsb/springer2/RNAseq_GATK_JGW/analysis/STAR/output_2nd_pass"
OUTPUT="$HOME/jsb/springer2/RNAseq_GATK_JGW/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[$SGE_TASK_ID -1]} \
RGPL=ILLUMINA \
RGPU=${RGPU[$SGE_TASK_ID -1]} \
RGID=${RGPU[$SGE_TASK_ID -1]} \
RGSM=$input

conda deactivate
```

After completion of the previous step, we will want to validate that the reads groups have been correctly set. When present, read groups are written in the header of BAM files. Thus, this simple UNIX trick allows one to quickly inspect the read groups associated to a BAM file:

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

To see read groups on all our BAM files, use something like:

```
for i in `ls *.bam | xargs basename -s '.bam'`;\
do samtools view -H $i.bam | grep '@RG'; done
```

```
@RG ID:C5NL3ACXX.1.CAGATC LB:A_CTL24 PL:ILLUMINA SM:SRR5487372 PU:C5NL3ACXX.1.CAGATC
@RG ID:C5NL3ACXX.1.TGACCA LB:A_MAP24 PL:ILLUMINA SM:SRR5487376 PU:C5NL3ACXX.1.TGACCA
@RG ID:C5NL3ACXX.3.TGACCA LB:B_CTL24 PL:ILLUMINA SM:SRR5487378 PU:C5NL3ACXX.3.TGACCA
@RG ID:C5NL3ACXX.3.GTGAAA LB:B_MAP24 PL:ILLUMINA SM:SRR5487382 PU:C5NL3ACXX.3.GTGAAA
```

3.3 Marking Duplicates Reads

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 offers the possibility to completely remove the duplicate reads and to make the distinction between optical and PCR duplicates. Here we will simply identified the duplicate reads in our .bam files before proceeding to the next step:
MarkDuplicates.sh:

```
#!/bin/bash
#$ -N MarkDuplicates
#$ -o logfile.$TASK_ID.log

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"
```

3.4 Splitting RNAseq reads

With the non-GATK alignment of raw sequences, the SplitNCigarReads step, (formerly called SplitNTrim step in previous versions of GATK workflows), is highly specific to RNAseq analysis. Reads that span splice junctions (for example reads that align over distinct exons) are split in smaller reads during the process and map at the right location in the alignments files.

3.4.1 Preparing Reference Genome Files

Until now, we have work with STAR reference genome which barely contains sequences that are transcribed in RNA. In contrast, the GATK SplitNCigarReads command requires a complete reference sequence as input. At this point, one should be advised to download the same version of the refence genome that was used to prepare the STAR genome.

Here we use the Bovine Genome Database https://bovinegenome.elsiklab.missouri.edu/ to get the UMD3.1 assembly.

```
wget http://128.206.12.216/drupal/sites/bovinegenome.org/\
files/data/umd3.1/UMD3.1_chromosomes.fa.gz
gunzip UMD3.1_chromosomes.fa.gz
```

The bovine genome database also contains several annotations files for the UMD3.1 assembly. We will use the Ensembl annotation file of protein and non-protein-coding genes when visualizing alignments with genome browsers at the end of this tutorial.

```
wget http://128.206.12.216/drupal/sites/bovinegenome.org/\
files/data/umd3.1/Ensembl79_UMD3.1_genes.gff3.gz
gunzip Ensembl79_UMD3.1_genes.gff3.gz
```

It is crucial that entries in the reference genome match the corresponding ones in the STAR genome. If you inspect the fasta headers of the UMD3.1_chromosomes.fa, you will notice that each fasta entry contains many fields (e.g. gnl , UMD3.1 Accession numbers):

```
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|GK000009.2 Chromosome 9 AC_000166.1
>gnl|UMD3.1|AY526085.1 Chromosome MT NC_006853.1
>gnl|UMD3.1|GK000030.2 Chromosome X AC_000187.1
>gnl|UMD3.1|GJ057137.1 GPS_000341577.1 NW_003097882.1
>gnl|UMD3.1|GJ057138.1 GPS_000341578.1 NW_003097883.1
...
```

In contrast, the chromosomes entries in the STAR genome are quite different:

```
cat Bos_taurus.UMD3.1.87_index/chrName.txt | head -n 34
```

SHOW THE RESULT

Therefore, the chromosome identifiers found in the *Bos taurus* UMD3.1 genome need to be changed to match their counterparts in the STAR genome. This could be achieved with UNIX SED:

```
sed -r s'/^>.+Chromosome\s+(\S+)\s+.+/>\1/' UMD3.1_chromosomes.fa > temp1.fa
grep ">" temp1.fa | head -n 40

# To deal with the unassigned scaffolds (accessions that begin with "GJ").
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
```

Note that we have rename the edited UMD3.1_chromosome assembly file into refGenome.fasta

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
```

On the other hand, use the GATK CreateSequenceDictionary tool to create the required .dict file:

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

3.4.2 Running SplitNCigarReads

Having prepared reference genome files and index, we are ready to lauch the following SplitNCigarRead step:
SplitNCigarReads.sh:

```
#!/bin/bash
#$ -N SplitNCigarReads
#$ -o SplitNCigarReads.$TASK_ID.log

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
```

3.5 Performing Base Quality Score Recalibration

Base Quality Score Recalibration (BQSR) is an optional, but highly recommended step, that is part of DNAseq and RNAseq GATK best practices workflows. This complex procedure, which involves binning and machine learning consist of re-evaluating all base quality scores assigned by sequencing machines which are prone to systematic technical errors [13]. Several features are taken into account, notably the reported quality score, the position within the read and the sequence context. This procedure lead to more accurate base calls, a situation that improve the accuracy of variant calling overall. More details about how the model is build and how new quality scores are computed can be found at https://gatk.broadinstitute.org/hc/en-us/articles/360035890531-Base-Quality-Score-Recalibration-BQSR-.

3.5.1 Using BaseRecalibrator

In the first key step a model of covariation is build with two components: the alignement given as input (BAM files) and a set of known variants (VCF file). The BaseRecalibrator will then produce a recalibration file which is used in the second key step of the procedure. To perform this task one would need a source of reliable variants in the investigated genome. With human data and other common model organisms, it would be easy to find such sources of variants, thanks to the existence of variation databases. Here we use a single VCF file containing the results of the BovineSNP50 genotyping BeadChip to train the model, but you can feed the model with several VCF files.

Before running the BaseRecalibrator command, index the SNP50 VCF file with:

```
gatk IndexFeatureFile -I snp50.vcf.gz
```

BaseRecalibrator_first_step.sh:

```
#!/bin/bash
#$ -N BaseRecalibrator
#$ -o BaseQualityRecalibration_$TASK_ID.log
```

```
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 ./known.vcf \
-O $OUTPUT/$input"_recal_data.table"

conda deactivate
```

3.5.2 Using ApplyBQSR to Adjust the Scores

In the second step of the BaseQuality recalibration process, the GATK ApplyBQSR command assigns new scores to each base based on the model produced in the first step and produces new BAM files that reflet these changes.
ApplyBQSR_second_step.sh:

```
#!/bin/bash
#$ -N ApplyBQSR
#$ -o ApplyBQSR_$TASK_ID.log

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
```

3.5.3 Comparing pre- and post-recalibration metrics

To visualize the effect of the procedure, one can build a seconde model and generate plots with data before and after recalibration [13] The GATK command AnalyzeCovariates was created for that purpose as in the following example:

```
gatk AnalyzeCovariates \
-before recal1.table \
-after recal2.table \
-plots AnalyzeCovariates.pdf
```

The Base recalibration being the final step in the data cleanup part of the workflow present here (figure 1), we are now ready for discovering variants from our analysis-ready RNAseq reads with the joint genoyping approach.

3.6 Joint Genotyping Variant Calling

3.6.1 Calling Variants Per-sample (GVCF mode)

In this step, the GATK HaplotypeCaller engine identifies candidate variation sites and record them in GVCF files. This is why this step has been called "GVCF workflow". GVCF files act as intermediate between analysis ready reads (BAM files) and the final joint analysis of multiple samples [8]. Such intermediate files facilitate the incremental discovery of variants and should be kept since they can be reused in novel rounds of joint genotyping analysis. Our listing to produce GVCF files for each sample reads as follow:
singleSampleGVCFcalling.sh:

```
#!/bin/bash
#$ -N GenotypeGVCF
#$ -o GenotypeGVCF.log

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 \
--dontUseSoftClippedBases true \
-R ./refGenome.fasta \
-I $BAMPATH/$input"_recal.bam" \
-O $OUTPUT/$input".g.vcf.gz" \
-ERC GVCF
```

```
conda deactivate
```

3.6.2 Merging of GCVF files

Merging of GCVF files has been revised recently by GATK experts by the addition of the novel GenomicsDBImport command. It resembles the former CombineGVCFs command, but promises better performances for parsing of VCF files and computing values stored in the VCF INFO fields(https://gatk.broadinstitute.org/hc/en-us/articles/360036883491-GenomicsDBImport). Interestingly, under the hood, this command make use of a novel data managament system called TileDB which is particluarly useful for representing huge sparse 2D arrays such as the genomic data found in VCF files [14].

The command requires a map files that list file paths of all GVCF files to be included in the resulting database. The -L option specifies the range genomic intervals on which the command operate and may be used several times. Since, we are interested to call variants on all bovine autosomes, we used it many times in the following listing:
GenomicsDBImport.sh

```
#!/bin/bash
#$ -N GenomicsDBImport
#$ -o GenomicsDBImport.log

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 28 \
-L 29 \
--sample-name-map $SAMPLES/"cohort_1.sample_map" \
--tmp-dir $HOME/jsb/tmp \
--reader-threads 5

conda deactivate
```

3.6.3 Joint Genotyping

The final step in this variant calling workflow is performed with the GATK GenotypeGVCFs command (https://gatk.broadinstitute.org/hc/en-us/articles/360037057852-GenotypeGVCFs). During this operation, all samples for which variant have been pre-called with HaplotypeCaller are jointly genotyped. This step could be rerun when new samples are available, allowing the so-called incremental discovery of variants.

The command take as argument a GenomicsDB workspace created by GenomicsDBImport, which can be seen as a sophisticated database that facilitate the parsing of the genomic data present in GVCF files. Alternatively, a single multi-sample GCVF file produced by CombineGVCFs (used after HaplotypeCaller) can also be given as input.
GenotypeGVCF.sh:

```
#$ -N GenotypeGVCF
#$ -o GenotypeGVCF.log

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
```

3.7 Querying VCF files

As mentionned earlier, BCFtools are optimized by design, to query and manipulate VCF files. Therefore, it is worth the pain to familiarize with these tools rather than using the plain VCF files with UNIX tricks. In addition, since BCFtools work well with compressed files it would be a waste of time to compress or extract VCF files not to mention the risk of corrupting or erasing a VCF file. Because one can easily erase a file by mistakes when attempting to redirect the result of querying and filtering commands, it is a good practice to keep a copy of the primary VCF output file (output.vcf.gz) in a safe place.

That said, the BCFtools command are highly versatile and can be used in several circumstances ranging from performing simples queries to applying complex filters. Here a few examples:

To get a list of all samples present in a VCF files:

```
bcftools query -l output.vcf.gz
```

To get useful metrics, like the sequencing depth or the ratio of transitions and transversions use the built-in stats command:

```
bcftools stats output.vcf.gz  > stats.txt
```

To quickly see the numbers of variants that fall in each category:

```
bcftools stats output.vcf.gz | grep -E "^SN"
```

To mask the VCF header and see variants from selected samples in a specific regions:

```
bcftools view -H -s SRR5487378,SRR5487390 -r 1:1-100000 output.vcf.gz
```

The query command make possible to fetch some VCF columns and manipulate them easily. For example, the command:

```
bcftools query -i 'QUAL>20 && INFO/DP>10' \
           -f'chr%CHROM pos:%POS Q:%QUAL DP=%INFO/DP\n' \
output.vcf.gz | head -n 2
```

give the following result:

```
chr1 pos:242647 Q:2073.95 DP=46
chr1 pos:242649 Q:2073.95 DP=46
```

3.8 Filtering variants

Contrarily to DNAseq workflows where sophisticated methods for filtering variants are available (e.g. VQSR and CNNScoreVariants), the current recommandation for RNAseq is to use hard filters as described in the canonical RNAseq Best practice RNAseq short variant discovery workflow [4]. The GATK and Picard tools have several commands to filter variants although the BCFtools filter and view commands can also be used. To perform hard filter, one could use the GATK VariantFiltration command (https://gatk.broadinstitute.org/hc/en-us/articles/360037269391-VariantFiltration) which is designed to filter variants on the basis of annotations present in INFO or FORMAT VCF fields.

Naturally, the extend of filtration steps that could be done depends on the nature of the downstream analysis that will be performed. Filtering variants is an active aera of research and the reader is invite to consult the GATK Best practices workflows documentation to be aware of all the novelties.

Among the numerous filtering that could be done, we present here a filtering step specific to RNAseq data that was seen in previous Best practices workflows. It consist of filtering out clusters of at least 3 SNPs that are within a window of 35 bases between them.

```
gatk VariantFiltration \
-R refGenome.fasta \
-V output.vcf.gz \
-O filtered.vcf.gz \
--cluster-window-size 35 \
--cluster-size 3
```

Notice that the GATK command writed the strings 'PASS' or 'SnpCluster' in the FILTER field (rather than the dot that was present in the orignal output VCF file) and added the following lines to the VCF header:

```
##FILTER=<ID=SnpCluster,Description="SNPs found in clusters">
##FILTER=<ID=PASS,Description="All filters passed">
```

Thus, to actually get rid off these variants, we can use the BCFtools view command with the –apply-filter option:

```
bcftools view --apply-filters .,PASS filtered.vcf.gz | bgzip -c > purged.vcf.gz
```

Note how the BCFtools are fully integrated with other UNIX commands. Here the filtered VCF files is given to the HSTlib bgzip program that compress it before that the UNIX file redirection operator write the result to a new file. You can validate that SNP clusters do not appear in the purged.vcf.gz with:

```
bcftools view purged.vcf.gz | grep SnpCluster
```

In DNA-seq, the generic current recommendation is to filter out variants based on Fisher Strand values (FS $>$ 60.0) and Quality by depth values (QD $<$ 2.0).

At first, we could be interested to know how many variants will be excluded with FS $>$ 60.0:

```
bcftools filter -i 'FS > 60' purged.vcf.gz | bcftools view -H | wc -l
2610
```

and how many variants will be excluded with QD $<$ 2.0:

```
bcftools filter -i 'QD < 2.0' purged.vcf.gz | bcftools view -H | wc -l
15089
```

To correctly translate the previous recommendation and combine the two criterions, one would be advised to use the logical '||' operator (OR) (rather than '&&' (AND)) because one will want to throw out any variants that fail at least one criterion:

```
bcftools filter -i 'FS > 60 || QD < 2.0' purged.vcf.gz | bcftools view -H | wc -l
17273
```

To actually remove these variants, simply use the -e option (–exclude) and create a new file:

```
bcftools filter -e 'FS > 60 || QD < 2.0' purged.vcf.gz | bgzip -c > newfile.vcf.gz
```

Another reasonable hard filtering that can be done would be to exclude calls with poor quality and low read depth:

```
bcftools filter -e "QUAL < 20.0 && INFO/DP < 10" output.vcf.gz > filtered.vcf.gz
```

A very intersting ressource to adequately applying other hard filters can be found at https://gatk.broadinstitute.org/hc/en-us/articles/360035890471-Hard-filtering-germline-short-variants.

3.9 Examining and Visualizing Alignment files

At a moment you will want to visualize alignments files and see the correspondance between called variants found in VCF files and the final alignments. This will allow you to detect potential misaligned regions or other artifacts. Tablet https://ics.hutton.ac.uk/tablet/download-tablet/ and the Integrated Genome viewver ??https://software.broadinstitute.org/software/igv/ are good choices for such tasks.

3.9.1 Tweaking the GFF3 file

If you are interested to add a genome track representing annotated features, the Ensembl UMD 3.1 GFF3 file downloaded previously (subsection x) is your best option. However, as previously, chromosome names in this file differ from those present in the STAR genome and consequently from those present in our alignment files. With the command below, one can see the number of features annotated for each distinct identifier present in the GFF3 file:

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

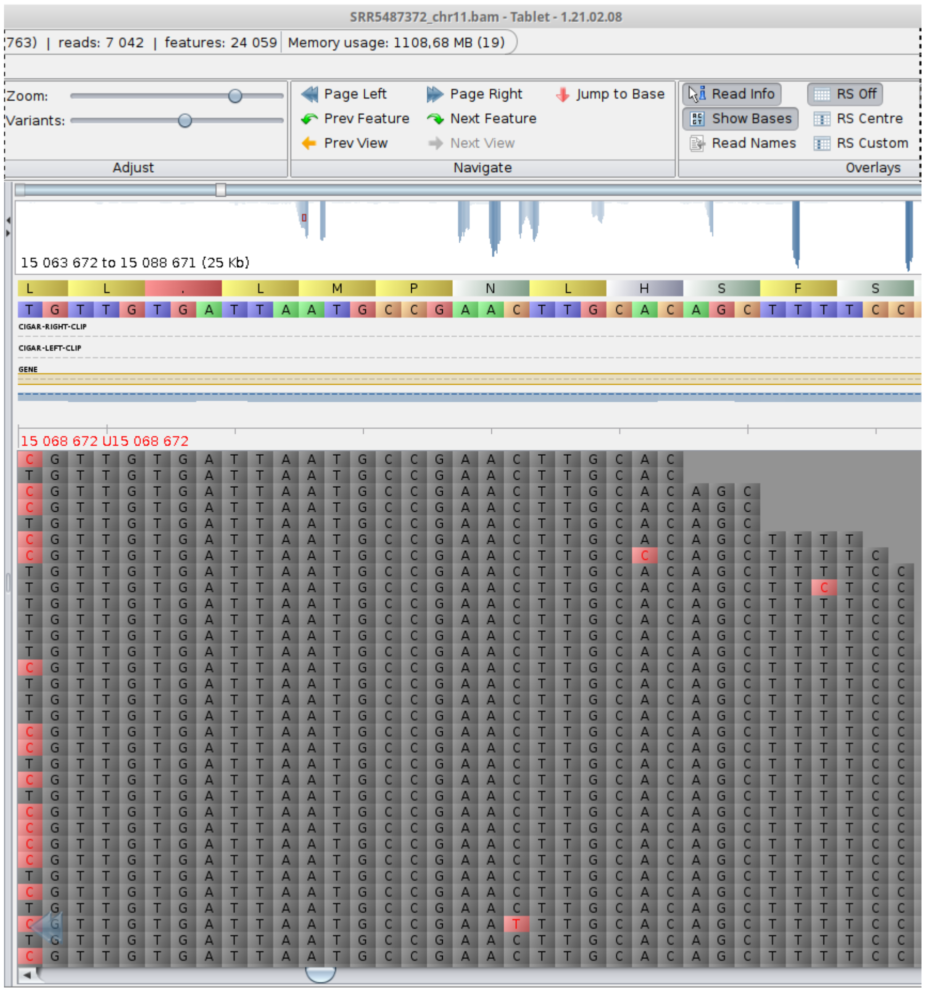
output:

```
21287 GK000001.2
18819 GK000030.2
26235 GK000002.2
29793 GK000003.2
...
14 GJ058430.1
31 GJ058425.1
...
23 GJ059509.1
3 GJ058729.1
3 GJ060027.1
3 GJ058256.1
```

Therefore,we need 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.gff3
```

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



SRR5487372_chr11.bam - Tablet - 1.21.02.08

763) | reads: 7 042 | features: 24 059 | Memory usage: 1108,68 MB (19)

15 063 672 to 15 088 671 (25 Kb)

15 068 672 U15 068 672

Figure 3: long desc.

3.9.2 Inspecting reads from sample SRR5487372

Here a screenshot of sample SRR5487372 in chromosome 11 between

bcftools view -s SRR5487372 -r 11:15063000-15068675 purged.vcf.gz

Figure 3 show an heterozygous call in a exonic region on chromosome 11.

In the upper panel we clearly can accumuilation of reads that correspond to exons from gene XX.

References

[1] 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.

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

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

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

[5] GATK. gatk4-rnaseq-germline-snps-indels, 2021. URL https://github.com/gatk-workflows/gatk4-rnaseq-germline-snps-indels.

[6] Jean-Simon Brouard, Flavio Schenkel, Andrew Marete, and Nathalie Bissonnette. The gatk joint genotyping workflow is appropriate for calling variants in rna-seq experiments. *Journal of animal science and biotechnology*, 10:44, 2019. ISSN 1674-9782. doi: 10.1186/s40104-019-0359-0.

[7] GATK. Germline short variant discovery (snps + indels), 2021. URL https://gatk.broadinstitute.org/hc/en-us/articles/360035535932-Germline-short-variant-discovery-SNPs-Indels-.

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

[9] Petr Danecek, James K. Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O. Pollard, Andrew Whitwham, Thomas Keane, Shane A. McCarthy, Robert M. Davies, and Heng Li. Twelve years of samtools and bcftools. *GigaScience*, 10, February 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab008.

[10] 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.

[11] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. Star: ultrafast universal rna-seq aligner. *Bioinformatics (Oxford, England)*, 29:15–21, January 2013. ISSN 1367-4811. doi: 10.1093/bioinformatics/bts635.

[12] GATK. Read groups, 2021. URL https://gatk.broadinstitute.org/hc/en-us/articles/360035890671-Read-groups.

[13] GATK. Base quality score recalibration, 2021. URL https://gatk.broadinstitute.org/hc/en-us/articles/360035890531-Base-Quality-Score-Recalibration-BQSR-.

[14] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. The tiledb array data storage manager. 10:349–360, 2016. ISSN 2150-8097. doi: 10.14778/3025111.3025117.