Preparation of alignments for variant calling with GATK: exercise instructions for BioHPC Lab computers

Data used in the exercise

We will use *D. melanogaster* WGS paired-end Illumina data with NCBI accessions SRR1663608, SRR1663609, SRR1663610, SRR1663611, corresponding to samples ZW155, ZW177, ZW184, and ZW185 respectively. To speed up the calculations, the data, originally at about 10 million read pairs per sample, has been down-sampled by 50%.

The exercise consists of several steps leading from Illumina reads to alignments ready for variant calling, as specified in GATK Best Practices. To call variants an all four samples, all these preparatory steps need to be performed for each of the four FASTQ file pairs. On a multi-CPU machine with large memory, such runs could be launched in parallel (for example, by manually submitting a given step in the background, each time for different sample). For the purpose of the exercise, please run the whole pipeline for at least one sample of your choice.

Log in to your workshop machine

The machine allocations are listed on the workshop website: https://cbsu.tc.cornell.edu/ww/machines.aspx?i=61.

Details of the login procedure using ssh or VNC clients are available in the document https://cbsu.tc.cornell.edu/lab/doc/Remote_access.pdf.

Use your **ssh client** with BioHPC Lab credentials to open an ssh session. If you wish, you can open multiple sessions to have access to multiple terminal windows (useful for program monitoring).

Alternatively, use the **VNC client** to open a VNC graphical session (you will need to first start the VNC server on the machine from "**My Reservations**" page reachable from https://cbsu.tc.cornell.edu after logging in to the website. To close the VNC connection, click on the "X" in top-right corner of the VNC window (but **DO NOT log out!**). This will ensure that your session (all windows, programs, etc.) will keep running so that you can come back to it by logging in again.

General tips

Examine all the provided shell scripts (`*.sh`), for example, opening them in a text editor (such as `nano` or `gedit`). Read explanatory comments. Notice the use of environment variables to simplify and generalize scripts. For example, following the definition of a variable `ACC`

```
ACC=SRR1663609
```

any time **$ACC** or **${ACC}** appears in the script, it will be interpreted as **SRR1663609**. Note the technique of breaking long lines into smaller pieces terminated with the "\" character. For bash this is still a long line, but easier to read for us.

Monitor the progress of your activities using the top command, preferably run in a separate window:

```
top –u <yourID>
```

This will show dynamically updated list of your processes, with the most active ones on top. Since both the GATK and PICARD tools are written in Java, the process you will see most will be Java virtual machine called **java**. In the alignment stage, the process to look for will be the BWA aligner called **bwa**. The absence of any active processes (consuming CPU time) on your top list will indicate completion (or crash) of any scripts you were running. Pay attention to memory usage (%MEM column) of different runs.

Peek into the log files. Each time a script is run, the screen output from all commands is saved into a log file (say, **script.log**). Although the messages written to that file may sound cryptic at times, they generally allow the user to figure out which stage of the calculation is running at the moment. It also contains useful timing information (start and end dates of individual stages, elapsed time, ETA time). To look into the log file, you can use any of the following commands

```
more script.log          (page through the file from the beginning)
tail -100 script.log     (display the last 100 lines of the file)
tail -f script.log       (continuously display incoming lines)
```

Of course, you can also look at the whole file by opening it in a text editor. Upon exit, **discard any changes** you may have inadvertently made.

Look into the working directory (/workdir<yourID>). As the run progresses, various intermediate files are being produced. Executing ls –al once in a while will allow you to see those files and how they increase in size.

If you can't see the expected output file even though it seems that the script has ended, it usually means that something went wrong. Examine the screen log file (e.g., open it in text editor) looking for error messages.

You can disconnect. If a step takes longer than you are willing to wait, you can disconnect from your VNC session (click on the cross in top right corner of the VNC window, but do **not** "Log Out"!). All your programs and windows will continue running and you can examine the results when you reconnect at a later time. Also, if you are working via ssh client (rather than VNC), you can safely log out of your ssh session as long as the script you are running was submitted in the **background** through **nohup** (as recommended throughout this exercise). The script will still be running (or will have finished) next time you log in to the machine.

Fetch input files and scripts to your local scratch directory

If not yet done, create your subdirectory in the scratch file system **/workdir**. In the following, we will assume the user ID **<yourID>** – please replace it with your own user ID.

```
cd /workdir
mkdir <yourID>
cd <yourID>
mkdir tmp
```

(The last line will create a temporary directory where Java will be instructed to store its scratch files.) Copy the exercise files from the shared location to your scratch directory (it is essential that all calculations take place here):

```
cp –R /shared_data/Variants_workshop_2015/* .
```

When the copy operation completes, verify by listing the content of the current directory with the command **`ls -al`**. You should see read files:

```
SRR1663608_thinned_1.fastq.gz
SRR1663608_thinned_2.fastq.gz
SRR1663609_thinned_1.fastq.gz
SRR1663609_thinned_2.fastq.gz
SRR1663610_thinned_1.fastq.gz
SRR1663610_thinned_2.fastq.gz
SRR1663611_thinned_1.fastq.gz
SRR1663611_thinned_2.fastq.gz
```

Check the file sizes – they should be around 410-430 MB. Along with the read files, several shell scripts **`*.sh`** are provided, corresponding to various stages of the pipeline. The subdirectory **`genome`** contains a FASTA file **`genome.fa`** with the *D. melanogaster* reference genome. The files **`*.vcf`** contain information about known variant sites – we will use them in the base quality recalibration stage.

Check the sequencing quality (optional)

This step summarizes the sequencing quality of the data. It is recommended to run this step before starting the assembly – it may help set the read trimming parameters for Trinity run.

```
cd /workdir/<yourID>
mkdir qcreport
fastqc -o qcreport *.fastq.gz
```

- **`-o qcreport`** : specify the output directory (./**`qcreport`**) where the QC reports will be stored, on directory per fastq file.
- All the fastq files should be specified, separated by space " ". The wildcard * also does the job.
- After it is done, you can use any sftp client (e.g., **FileZilla**) to copy the **`qcreport`** directory to you laptop computer, and open the **`fastqc_report.html`** file in each subdirectory with a web browser. If you are working in graphical environment (i.e., via VNC), you can launch the Firefox browser directly on the Linux workstation and navigate to **`fastqc_report.html`** files.

Prepare reference genome

The file **genome.fa** in subdirectory genome is the reference we will be aligning the reads to. Before starting the pipeline, the genome needs to be indexed for the BWA aligner. Also, length information for all chromosomes needs to be summarized.

All this can be done by running the shell script **prepare_genome.sh**. Run the script as follows:

```
nohup ./prepare_genome.sh >& prepare_genome.log &
```

The script will run the background and should take a few minutes to complete (use **top –u <yourID>** in a separate window to monitor your processes). After it completes, list the content of subdirectory **genome**. The files **genome.fa.fai** and **genome.fa.dict** are simple text files summarizing chromosome sizes and starting byte positions in the original FASTA file. The other files constitute the BWA index.

Align reads to reference

The script **aln_bwa.sh** will start the **bwa** aligner for accession **SRR1663609** (sample ZW155). Look at the script (e.g., with the nano editor). In particular, note the definition of read group in bwa command line (bwa will insert it into the SAM file it generates). In our exercise, each sample is sequenced once on one Illumina lane.

Start the script (in **/workdir/<yourID>**):

```
nohup ./bwa_aln.sh >& bwa_aln_SRR1663609.log &
```

The program will run in the background, saving any screen output to the log file we decided to name **bwa_aln_SRR1663609.log**. Approximate run time: **10 minutes**.

The result will be the file **SRR1663609.sam**, describing the alignments in SAM format. This file is a text file – you can take a quick look at it using standard Linux commands, like **more**, **head**, **tail**. It may, however, be too large to be opened with **nano** or other text editors.

If you wish, repeat the run with different sample(s) (it will come in handy in the next session, when we will be calling variants from multiple samples). To do this, edit **bwa_aln.sh**, and change the SRR accession and the corresponding sample name where needed (don't forget the read group declaration!), and re-run the script.

Convert to BAM, sort, and mark duplicates

The SAM file obtained previously will now be sorted and converted to BAM format (same as SAM, but compressed to save space on disk). Any duplicate fragments will then be marked to be ignored in downstream analysis. To do this, launch the script **sort_dedup_index.sh**:

```
nohup ./sort_dedup_index.sh SRR1663609 >& sort_dedup_index_SRR1663609.log &
```

The script takes one argument (the SRR accession). To re-run it for a different accession, just change the argument in the command above – there is no need to change anything in the script itself. However, edit the script and examine the syntax of the commands and any informative comments it may contain. The run for one sample will take approximately **10 minutes**. Examine the log file for any errors and detailed timing information. It is actually convenient to monitor the log file during the run (**`tail –f sort_dedup_index_SRR1663609.log`**).

The result of the run will be the file **`SRR1663609.sort.dedup.bam`**, accompanied by the index file **`SRR1663609.sort.dedup.bai`**.

Check the alignment stats summary of the obtained file, running the **`samtools`** command

```
samtools flagstat  SRR1663609.sort.dedup.bam
```

Can you tell how many reads have been mapped? How many have been marked as duplicate?

Local realignment

To identify regions containing indels and regions likely to harbor a hidden indel, and then collectively realign all reads mapped to these regions, use the script realign.sh. Looking at this script, you will notice the last command – an invocation of PICARD's **`FixMateInformation`** program – which re-syncs mate pair coordinates that may have changed as a result of the realignment. The command (entered, as usual, from **`/workdir/<yourID>`**)

```
nohup ./realign.sh SRR1663609 >& realign_SRR1663609.log &
```

will launch the realignment process for sample **`SRR1663609`**, producing the file **`SRR1663609.sorted.dedup.realigned.fixmate.bam`** (with the corresponding index). An intermediate file **`SRR1663609.sorted.dedup.realigned.bam`** (containing realigned reads with incorrect mate pair information) may be discarded. Estimated run time: **20 minutes**.

Base quality score recalibration

The script recalibrate.sh first collects the mismatch data from alignments and then uses this information to re-calibrate base quality scores of reads in the BAM file obtained in the previous step. It is important that the mismatch data is collected from a BAM file in which reads have been locally re-aligned around indels, since these alignments are less likely to contain false mismatches. Any well-known variant positions should be excluded from the mismatch count. We have such known variant position sets for the four chromosomes: chr2L, chr2R, chr3L, and chr3R (in files **`chr*.vcf`** provided with the exercise data). These are specified in the **`BaseRecalibrator`** command (using **`–knownSites`** option). **`BaseRecalibrator`** is only using alignments from these four chromosomes (as specified by the **`–L`** options). This way, any unknown but true variant sites of other chromosomes will not skew the recalibration database. The database is then applied in the second step to the full alignment file.

To launch the script for sample **`SRR1663609`**, execute

```
nohup ./recalibrate.sh SRR1663609 >& recalibrate_SRR1663609.log &
```

The output consists of two files: the recalibration table **SRR1663609.sorted.dedup.realigned.fixmate.recal_data.table** and the recalibrated BAM file **SRR1663609.sorted.dedup.realigned.fixmate.recal.bam**. Estimated run time: **35 minutes**.

Create "before/after" base quality plots (optional)

For illustration purposes, the recalibration procedure may be repeated using the already recalibrated file (here: **SRR1663609.sorted.dedup.realigned.fixmate.recal.bam**) as input. The new recalibration table obtained in this way is expected to show little difference between the new empirical base quality scores and the old (already recalibrated) ones. This new recalibration table can be used along with the old one (obtained in previous step) to illustrate the effect of recalibration on base scores within different covariate bins. Running the script **recal_plot.sh**

```
nohup ./recal_plot.sh SRR1663609 >& recal_plot_SRR1663609.log &
```

will result in a new recalibration table **SRR1663609.sorted.dedup.realigned.fixmate.postrecal_data.table** and the "before/after" plot in pfd format: **SRR1663609.sorted.dedup.realigned.fixmate.recal_before_after.pdf**.

Estimated run time: **15 minutes**.

Visualize the alignments using IGV

You need to be connected to the machine with VNC. In a terminal window, enter

```
igv
```

The IGV window will appear in a few moments. In the navigation bar, go to Genomes and select **Load Genome from File**. Browse to **/workdir/<yourID>/genome** and select the file **genome.fa**. I the chromosome selection dropdown (which initially says **All**) you should see *D melanogaster* chromosome names. Now load one of the BAM files created during the exercise: go to **File → Load from File**, browse to **/workdir/<yourID>**, and select one of the BAM files. Select one of the chromosomes and zoom in (using the slider in top right corner) enough to see alignments.

IGV has plenty of display options, some available in **View** menu tag, some showing up upon a right click within the display window. Try to familiarize yourself with the program's capabilities. For example, try different read coloring schemes based on various properties of aligned pairs (right click within window, then select **Color alignments by**…). Notice that left-clicking on (or just hovering over – it is configurable) a read will display detailed information about it, as found in the BAM file. You may find it helpful to use IGV's **Help** link to learn more.

You can load one or more additional BAM files if you wish. It is especially instructive to compare BAM files before and after local realignment step (**SRR1663609.sorted.dedup.bam** and **SRR1663609.sorted.dedup.realigned.fixmate.bam**, respectively). Can you identify reads

that have been re-aligned? **Hint**: such reads may be rare and hard to spot. Fortunately, the realigner leaves its "mark" in the resulting BAM file by introducing a tag **OC** (for "old CIGAR") into the BAM record of each read whose alignment was altered. To look for such reads, simply run the command

```
samtools view SRR1663609.sorted.dedup.realigned.fixmate.bam | grep "OC:Z" | more
```

(it is important to pipe the lengthy output through the more pager tool to prevent it from scrolling too fast). Write down the chromosomes and starting alignment positions of a few such reads, and navigate to those coordinates in IGV.