Variant calling with GATK: exercise instructions for BioHPC Lab computers

Introduction

In this exercise we will practice calling and processing variants from alignments of *D melanogaster* data (Grenier JK, Clark AG, et al. (2015) Global Diversity Lines - A five-continent reference panel of sequenced Drosophila melanogaster strains. G3 (Bethesda). 2015 Feb 11. pii: g3.114.015883) prepared during the previous session of the workshop. Initial step of the procedure is time-consuming (at least 2 hours of compute time). If you wish, you can skip this step and use the pre-computed intermediate files instead (see step instructions for details).

Log in to your workshop machine

The machine allocations are listed on the workshop website:
https://cbsu.tc.cornell.edu/ww/machines.aspx?i=674.

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.

Scripts and input data

First, copy the shell scripts to be used in the exercise to your local scratch directory **`/workdir/<yourID>`** (you used this directory in previous exercise, so it should already be there):

```
cd /workdir/<yourID>
cp /shared_data/Variants_workshop_2016/part2/*.sh .
```

Also, make sure that the directory exists (if not – create it) – the scripts ask java to store its scratch files there. After the previous exercise, you should also have a directory **`/workdir/<yourID>/genome`** containing reference genome sequence and index files.

In previous exercise, you were supposed to obtain at least one of the de-duped, re-aligned, and recalibrated BAM files:

```
SRR1663608.sorted.dedup.realigned.fixmate.recal.bam
SRR1663609.sorted.dedup.realigned.fixmate.recal.bam
SRR1663610.sorted.dedup.realigned.fixmate.recal.bam
SRR1663611.sorted.dedup.realigned.fixmate.recal.bam
```

along with the corresponding index (*.bai) files. For this exercise, you will need all of these BAM files in your work directory. You can fetch all of them (along with their index file **`*.bai`**) from the workshop shared directory:

```
cp /shared_data/Variants_workshop_2016/part2/*.ba? /workdir/<yourID>
```

Run GATK HaplotypeCaller on individual samples

In this step, we will run the **`HaplotypeCaller`** on our BAM files to produce genotype likelihood information for each sample for each locus in the genomic region of interest. Typically, this region would be the whole genome. To save time, we will concentrate on one chromosome, **chr2R** (see the **`–L`** option in the **`hc.sh`** script). To launch the calculation for one of the sample (e.g., **`SRR1663608`**), type

```
nohup ./hc.sh SRR1663608 >& hc_SRR1663608.log &
```

The screen output from the command will be written to the log file specified above. The expected result will be the file **`SRR1663608.g.vcf`**. containing the intermediate genotyping data for this sample.

The estimated run time of this step is 1 hour, and it has to be repeated for the other 3 samples. In real life, calculation for different samples would be run concurrently as different processes on the same multi-core machine, or on separate machines. It is also possible to parallelize the calculation over genomic coordinate, i.e., run separate jobs per sample per (a piece of) a chromosome (which can be specified with the **`–L`** option of **`HaplotypeCaller`** – see **`hc.sh`** script).

For the purpose of the exercise, we would recommend that you run this step for at least one of the samples (e.g., the one for which you prepared the BAM file in last session), and then fetch the other (pre-computed) **`*.g.vcf`** files (and corresponding index files **`*.g.vcf.idx`**) from the shared workshop directory, e.g.,

```
cp /shared_data/Variants_workshop_2016/part2/SRR1663610.g.vcf* .
```

and similarly for other files. If you are short of time, you can simply skip this calculation and fetch all the ready-made **`*.g.vcf`** files. In any case, please examine the **`hc.sh`** script.

Once the **`*.g.vcf`** files are in your work directory, examine them (e.g., open in **`nano`** text editor – possible for short test files like these) and confront with the gVCF format description at https://www.broadinstitute.org/gatk/guide/article?id=4017.

Joint variant calling with GenotypeGVCFs

The intermediate, sample-level files **`*.g.vcf`** will now be used to call variants jointly on all four samples. The corresponding GATK command can be found in the script **`joint_call_from_gVCF.sh`**. Run the script:

```
nohup ./joint_call_from_gVCF.sh >& joint_call.log &
```

As usual, the script will run in the background, saving all screen output to the log file. The script takes no arguments, since all the necessary information (sample IDs) is hard-coded in the script explicitly. The output (as specified in the GATK command line) is the file **`hc.chr2R.vcf`**, containing the raw (i.e., not yet filtered or recalibrated) variant calls for our 4-sample "population". Open this file in a text editor and examine its content. **Estimated run time: 5-6 minutes**.

Joint variant calling using UnifiedGenotyper

UnifiedGenotyper is another GATK tool for joint variant calling. It does not perform local re-assembly of haplotypes and operates on a site-by-site basis instead. As a result of this, it is faster than **HaplotypeCaller** approach, although generally less accurate.

UnifiedGenotyper takes de-dupped, re-aligned and recalibrated BAM files as input. You will have to have all 4 of these BAM files present in your scratch directory (copy them from the workshop shared space, as described above).

To run the tool, simply execute the script **`ug.sh`**:

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

The resulting file, **`ug.chr2R.vcf`**, will contain a set of variants produced. Estimate run time: **10 minutes**.

Joint variant calling using FreeBayes

FreeBayes is a variant-calling program by Erik Garrison et al., https://github.com/ekg/freebayes. It is independent from GATK. Similarly to GATK's **HaplotypeCaller**, **FreeBayes** uses haplotype-based approach to variant detection, although implemented differently.

The input for **FreeBayes** consists of alignment BAM files for all samples involved. Unlike GATK, the **FreeBayes**-based pipeline does not require extensive preparation of alignments, such as local re-alignment or base score recalibration. However, since such pre-processing won't hurt, we can use our processed BAM files obtained in the previous session (you can copy those from the workshop shared space, as described above). Assuming all BAM files are in place, call variants using the script **`fb.sh`**:

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

The result will be the variant file **`fb.chr2R.vcf`**. **Estimated run time: 20 minutes**.

Examining the resulting VCF file, notice that the parameters in the ANNOTATION field generated by **FreeBayes** are generally different than those emitted by GATK callers.

Filter variants with VariantFiltration

The VCF files obtained in previous steps are raw results, likely to contain a lot of false positives, depending on the stringency options used in calling. Since the calling steps are time-consuming, it is generally advisable to set these options to emit an inclusive set of variants, and then filter this set over various parameters, such as those recorded in the INFO field of a VCF file. In GATK, the option **`-stand_emit_conf`** controls the lower threshold on the quality (the QUAL field of VCF) of variants to be output. This option should be set to some low value (e.g., 5).

Filtering of the raw set of variants can be accomplished using many different tools. In a lot of cases, one can simply utilize standard Linux text parsing tools, like **grep**, **awk**, or **sed**. For example, to extract a subset of variants with QUAL greater than, say, 60, from raw **`hc.chr2R.vcf`** we could use the following commands:

```
grep ”#” hc.chr2R.vcf > hc.chr2R.qual60.vcf
grep –v ”#” hc.chr2R.vcf | awk ’{if($6>60) print}’>> hc.chr2R.qual60.vcf
```

The first command extracts the VCF header lines (containing “#”) into a new VCF file, while the second command processes the non-header lines, appending them to the new file only if the sixth column (that’s where QUAL is) is above 60.

GATK offers a tool called **VariantFiltration**, which allows more complex filtering patterns. An example script using this tool is called **`filter_vcf.sh`**. As you examine this script, you will notice that different filtering criteria are applied to SNPs than for indels. To accomplish this, SNPs and indels are first extracted to separate files, these files are filtered, and then the SNP and indel filtered files are merged back into a single filtered file. To run the filtering script, enter

```
      nohup ./filter_vcf.sh hc.chr2R >& filter_vcf.log &
```

(note that we are supplying the prefix of the VCF file name, i.e., *without* the **`.vcf`** extension as argument). The filtered VCF file will be called **`hc.chr2R.filtered.vcf`** (the corresponding index file **`*.vcf.idx`** will also be created). Other intermediate files (with separate SNPs, indels, filtered and not, along with their indexes) will also be produced – these may be deleted.

Examine the filtered VCF file. Notice the change in the FILTER field. Instead of dot “.” (no filtering information), this field will now contain flags **PASS** (variants which passed the filter) and **my_snp_filter** or **my_indel_filter** (both these strings were defined in filtering command) – marking variants which failed the respective filters. Note that **no variant has been removed from the file**. The ones that failed filtering are just marked as such.

Estimated run time: **3 minutes**.

Variant score recalibration

Variant Quality Score Recalibration (VQSR) is a procedure recommended by Broad to be used instead of the hard (threshold-based) filtering in cases when a large, verified set of high-quality SNPs and/or indels is available. These reliable sets of variants can be used to train a machine learning model (Gaussian mixture model) using various parameters (mainly from the ANNOTATIONS field) as attributes.

Once the model is trained, it is used to assign another score (called VQSLOD) to each unknown variant (this new score computed by running this variant's ANNOTATIONS parameters through the model). The new score is recorded as another parameter in the ANNOTATIONS field. A VCF file "recalibrated" this way may then be filtered over VQSLOD (the higher the value, the better the variant is) rather than through the application of often quite arbitrary hard thresholds.

The variant call error models for SNPs and indels are different, therefore VQSR is applied separately to these two kinds of variants using different training sets. In our example, we will use the known variants from the file **`chr2R.vcf`** (the same one we used for base score recalibration; it contains mostly SNPs) as a training set for recalibrating **`hc.chr2R.vcf`**. The script executing the procedure, called **`vqsr.sh`**, consists of two GATK commands. The first invokes the model training, the second applies the model to all variants in the input VCF file and produces the VQSLOD scores. After running the script

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

you will notice several output files with a string **`vqsr`** in file names. The final output file **`hc.chr2R.vqsr_snps_raw_indels.vcf`** will contain SNP variants with VQSLOD score appended (and still raw indels). In the FILTER field of this file, you will notice flags like **PASS** or **VQSRTrancheSNP99.00to99.90**. The flag **PASS** means that the VQSLOD score of the variant is high enough to reproduce at most top 99% of training variants (see parameter **`--ts_filter_level`** which sets this threshold). The flags **`VQSRTrancheSNP99.00to99.90`** or **`VQSRTrancheSNP99.90to100.00`** indicate lower VQSLOD, i.e., in a higher sensitivity tranche, such that more training variants (possibly with more false positives) would be included. The tranches are defined at the training model stage.

Estimated run time: **5 minutes**.

Basic stats and comparison of variant sets

Using Linux commands

Given a VCF file, its simplest properties may be obtained by running standard Linux text parsing tools. For example, to get the number of variants in a file, run the following:

```
grep –v "#" hc.chr2R.vcf | grep wc –l
```

(**`grep`** filters out the header lines and pipes its output into **`wc –l`** which counts the remaining lines and displays the result on screen). To extract sites located between positons 10000 and 20000 on chromosome chr2R and save them in a file, simply run

```
grep –v "#" hc.chr2R.vcf | awk '{if($1=="chr2R" && $2 >=10000 && $2
<=20000) print}' > extracted_records
```

(note that in this case the chromosome condition **`$1=="chr2R"`** is not really needed, because our VCF file only contains data for chr2R, however, it would be necessary for a more general input). To quickly find out how many variants passed filtering, simply type

```
awk '{if($7=="PASS") print}' uc.chr2R.filtered.vcf | wc -l
```

More complex analysis and operations on VCF files can be accomplished using specialized software tools, such as those contained in GATK package and those from the **vcftools** package (independent from GATK).

Using GATK's VariantEval

GATK offers an interesting function, **VariantEval**, to summarize various statistics of a variant set and compare it to another variant set obtained from the same data, but with a different method, for example. A script **`var_eval.sh`**, based on this function, will compare any two VCF files:

```
./var_eval.sh hc.chr2R ug.chr2R >& var_eval.log &
```

will generate a file **`hc.chr2R.ug.chr2R.comp.gatkreport`** with the result of the comparison. This file has long lines and is best viewed in Excel (after being transferred to your laptop). In particular, the first few lines show how many variant sites are shared between the two files, how many are "new" (i.e., absent in the comparison file), and what is the concordance rate in the shared variant set (i.e., what fraction of variants have the same alleles and genotypes).

Run the script for various pairs of VCF files obtained in previous steps. Are the variants from **HaplotypeCaller** and **UnifiedGenotyper** similar to each other? How about the ones from **FreeBayes**?

Using vcftools

vcftools (A. Auton, A. Amrcketta, http://vcftools.sourceforge.net/) is a popular toolkit for analyzing and manipulating VCF files. Here are some usage examples (try them on your VCF files):

Obtain basis VCF statistics (number of samples and variant sites):

```
vcftools --vcf hc.chr2R.vcf
```

Extract subset of variants (chromosome chr2R, between positions 1M and 2M) and write tem a new VCF file

```
vcftools –vcf hc.chr2R.vcf --chr chr2R --from-bp 1000000 --to-bp
2000000 --recode –recode-INFO-all -c > subset.vcf
```

Get allele frequencies for all variants and write them to a file

```
vcftools --vcf hc.chr2R.vcf --freq -c > hc.chr2R.freqs
```

Compare two VCF files (will print out various kinds of compare info in files **`hc.ug.compare.*`**):

```
vcftools --vcf hc.chr2R.vcf --diff ug.chr2R.vcf --out hc.ug.compare
```