A roadmap of whole exome sequencing (WES) data analysis

Yufeng He (yufenghe@stu.pku.edu.cn), Zexian Zeng(zexianzeng@pku.edu.cn)
Center for life sciences & Center for Quantitative Biology
Academy for Advanced Interdisciplinary Studies, Peking University
Lab website: http://cqb.pku.edu.cn/zenglab/

> Try to evaluate the bioinformatics work objectively, instead of by the impact factor of the journal the study is published.
>
> ---Shirley X. Liu' blog

Preparations of coding

1. log in your account on the server by

   `ssh -i {Path_to_your_private_key_file} -p 16122 {Your_username}@162.105.160.16`

   - If the system of your laptop is windows, you should download and install MobaXterm
   - If you are using MacOS, you can directly use your terminal.

2. Set up your environment variables

   - Make sure you are at your root directory after logging in via `pwd`
   - Make a new file namely ".bashrc" via `touch .bashrc`
   - Writing the path of conda into your environment variable file by `vim .bashrc` and copy the following contents into it:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/lustre1/share/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/lustre1/share/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/lustre1/share/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/lustre1/share/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

   - Type `:` and then `wq!` . This will save the changes of .bashrc file.
   - Finally, use `source .bashrc` to activate your environment variables.
   - Simply type `conda` and you'll see the usage of conda. It works!!!

3. We have installed multiple software that you will use in this course in the miniconda. Make sure all the software is ready via typing them one by one:

```
bwa
fastp
samtools
gatk
```

If you see no errors and it returns the usage of each tool, congratulations!!!

You need to enter the virtual environment by typing `conda activate wes`

Pre-class Assignments

1. Make sure you can log in your account before the class. (10') If you have any questions, feel free to ask any teaching assistant in WeChat group.
2. How many kinds of operating system do you know? What are their pros and cons? (10')
3. Find the document of common linux commands and see how they work. (40')
   `ls` `cd` `mkdir` `grep` `find` `mv` `cat` `top` `cat` `free` `exit` `tar` `unzip` `pwd` `echo` `touch` `cp` `vim` `rm` `scp`
4. Preview the tools that we will use to analyze WES data and explain how we can use these commands and how they work. (40')

Data

All the WES data that we will use in this class can be found at

`/lustre1/share/data/OC_WES`

You can use `cd` to get into this directory, and you will find that there are two sets of WES data from the samples of ovarian cancer (OC) and PBMC from one patients. In each directory, you can find 2 files storing biological sequences with extension `.fastq` or `.fq` in FASTQ format. They are also compressed with GZIP, so the suffix will be `.fastq.gz` or `.fq.gz`.

Now, we have prepared everything that will be used in our class.

Reference genome

All the reference genome can be found at

`/lustre1/share/references`.

---

1. Deal with FASTQ files.

Usually, you will get a FASTQ file compressed in GZIP format. But if you get one without GZIP compression, it is recommended to compress it via `gzip {Your_filename.fq}`.

Let's check the contents of the file via

`less {fastq_filename}.fq.gz`.

Type `:` and then `q` to exit.

You will see something like

```
@LH00380:102:2252H7LT4:2:1101:40943:1028 1:N:0:CCTCGAAT+TATGGCAC
GTCTTTGAAAAAAAATTTTGATTTGCTTTTCTGTATTTCTTTTTCAACTTTTAATCTATATTCCCAACTCTAATTCATTCTGTAATAATGTTAACCCCCAAAATTGTATGAACTGTCTTATTTGAATCA
+
IIIIIII9IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII-IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

@LH00380 --- the unique instrument name

102 --- the run id

2252H7LT4 --- the flowcell id

2 --- flowcell lane

1101 --- tile number within the flowcell lane

40943 --- 'x'-coordinate of the cluster within the tile

1028 --- 'y'-coordinate of the cluster within the tile

1 --- the member of a pair, 1 or 2 (paired-end or mate-pair reads only)

N --- Y if the read is fileterd, N otherwise

0 --- 0 when none of the control bits are on

CCTCGAAT+TATGGCAC --- index sequence

GTCTTTGAAAAAAAA.....TGCCTGAACTG --- sequence (A, C, G, T, N)

IIIIIII9II.....I9IIIII --- phred quality score

Quality scores started as numbers (0-40) but have since changed to an ASCII encoding to reduce filesize and make working with this format a bit easier, however they still hold the same information. **ASCII codes** are assigned based on the formula found below.

Formula: score + offset => look for American Standard Code for Information Interchange (ascii) symbol
Two variants: offset=64(Illumina 1.0-before 1.8); offset=33(Sanger, Illumina 1.8+).
A quality score is typically: [0, 40]

```
(33):  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
(64):  @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
```

Sanger, Illumina v1.3 to 1.7 (ASCII_BASE=64)

Q	ASCII	P	Q	ASCII	P	Q	ASCII	P	Q	ASCII	P
1	A	0.79433	12	L	0.06310	23	W	0.00501	34	b	0.00040
2	B	0.63096	13	M	0.05012	24	X	0.00398	35	c	0.00032
3	C	0.50119	14	N	0.03981	25	Y	0.00316	36	d	0.00025
4	D	0.39811	15	O	0.03162	26	Z	0.00251	37	e	0.00020
5	E	0.31623	16	P	0.02512	27	[	0.00200	38	f	0.00016
6	F	0.25119	17	Q	0.01995	28	\	0.00158	39	g	0.00013
7	G	0.19953	18	R	0.01585	29	]	0.00126	40	h	0.00010
8	H	0.15849	19	S	0.01259	30	^	0.00100			
9	I	0.12589	20	T	0.01000	31	_	0.00079			
10	J	0.10000	21	U	0.00794	32	`	0.00063			
11	K	0.07943	22	V	0.00631	33	a	0.00050			

Illumina v1.8 and later (ASCII_BASE=33)

Q	ASCII	P	Q	ASCII	P	Q	ASCII	P	Q	ASCII	P
1	"	0.79433	12	-	0.06310	23	8	0.00501	34	C	0.00040
2	#	0.63096	13	.	0.05012	24	9	0.00398	35	D	0.00032
3	$	0.50119	14	/	0.03981	25	:	0.00316	36	E	0.00025
4	%	0.39811	15	0	0.03162	26	;	0.00251	37	F	0.00020
5	&	0.31623	16	1	0.02512	27	<	0.00200	38	G	0.00016
6	'	0.25119	17	2	0.01995	28	=	0.00158	39	H	0.00013
7	(	0.19953	18	3	0.01585	29	>	0.00126	40	I	0.00010
8	)	0.15849	19	4	0.01259	30	?	0.00100	41	J	0.00008
9	*	0.12589	20	5	0.01000	31	@	0.00079			
10	+	0.10000	21	6	0.00794	32	A	0.00063			
11	,	0.07943	22	7	0.00631	33	B	0.00050			

Many programs require the FastQ format, implying that they will use the quality score in a particular part of the analysis. Common uses are to filter bases or entire reads if a particular quality threshold isn't met.

You can count the number of sequences like this:

`zcat {fastq_filename}.fq.gz | echo $(( wc -l /4))` .

You can also count how many times appear an specific subsequence via:

`zgrep -c 'ATGATGATG' {fastq_filename}.fq.gz` .

Sometimes we will be interested in extracting a small part of the big file to use it for testing our processing methods, ex. the first 1000 sequences (4000 lines):

`zcat {fastq_filename}.fq.gz | head -4000 | gzip > {Path_to_save}/{filename}.fq.gz`

2. Quality control

Before using fastp, you need to confirm you are at your root folder via `pwd` . If it outputs the path like `/home/{your name}` , continue the following tutorial. Let's make a new folder `analysis` and enter it:

`mkdir analysis && cd analysis`

Ususally, we store the output files in different files. So, we make a sub-folder to store the `FASTQ` files after quality control:

`mkdir 0_preprocess && cd 0_preprocess`

Now, we make two folders for OC and PBMC, repectively.

`mkdir OC && mkdir PBMC`

Fastp is developed in C++ with multithreading supported and designed to preprocess FASTQ files including quality profiling, filtering, triming, visualization and so on. To get help of fastp, you can simply type

`fastp` or `fastp -?` or `fastp --help`

As you can see, there are many options. We will only use the simple usage

`fastp -i in.R1.fq.gz -I in.R2.fq.gz -o out.R1.fq.gz -O out.R2.fq.gz`.

Let's firstly enter `OC` and see the current working directory:

`cd OC && pwd`

Now, you should be at a path like `/home/teach31_pkuhpc/analysis/0_preprocess/OC` (my account name is teach31_pkuhpc, you should change it to your account name in the following tutorial).

This is the last reminding of adding **path** before your file name. So, it should be like:

```
fastp -i /lustre1/share/data/OC_WES/OC/OC_R1.fq.gz -I /lustre1/share/data/OC_WES/OC/OC_R2.fq.gz \
-o /home/teach31_pkuhpc/analysis/0_preprocess/OC/OC_R1.fq.gz \
-O /home/teach31_pkuhpc/analysis/0_preprocess/OC/OC_R2.fq.gz
```

You should see the output `.fq.gz` files. Then we should go to `PBMC` directory and do the same:

`cd .. && cd PBMC && pwd`

```
fastp -i /lustre1/share/data/OC_WES/PBMC/blood_R1.fq.gz -I /lustre1/share/data/OC_WES/PBMC/blood_R2.fq.gz \
-o /home/teach31_pkuhpc/analysis/0_preprocess/PBMC/blood_R1.fq.gz \
-O /home/teach31_pkuhpc/analysis/0_preprocess/PBMC/blood_R2.fq.gz
```

Then you will find the clean FASTQ file in the output directory. Note that this is a simplified version. You should explore more on other options about adapter trimming, duplication, polyG tail trimming and other filtering options.

3. Alignment

In this part, we need to use `bwa` and `samtools` to align sequences with reference genome.

First, we go back the `analysis` folder:

`cd ../../`

we use `1_alignment` to store the outputs

`mkdir 1_alignment && cd 1_alignment && mkdir OC && mkdir PBMC`

BWA-MEM is one of three algorithms in BWA and is designed for longer sequences ranged from 70bp to a few megabases. BWA-MEM is faster and more accurate, and is generally recommended. We can use BWA-MEM via:

```
bwa mem -t 4 -M -R '@RG\tID:2\tPL:illumina\tSM:OC' \
/lustre1/share/references/hg38.fa \
/home/teach31_pkuhpc/analysis/0_preprocess/OC/OC_R1.fq.gz \
/home/teach31_pkuhpc/analysis/0_preprocess/OC/OC_R2.fq.gz \
> /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.sam
```

The -t option specifies the number of threads to use, which can speed up the alignment process.

The -M option marks shorter split hits as secondary, which is useful for compatibility with certain downstream tools.

The -G option is not necessary in most cases. It points a read group split by '\t' with the read group identifier (ID), platform (PL), sample (SM), PU (Platform Unit) and LB (DNA preparation library identifier). Not all read group fields are required.

It will take some time for `bwa` to finish alignment. The other one for PBMC would be:

```
bwa mem -t 4 -M -R '@RG\tID:2\tPL:illumina\tSM:blood' \
/lustre1/share/references/hg38.fa \
/home/teach31_pkuhpc/analysis/0_preprocess/PBMC/blood_R1.fq.gz \
/home/teach31_pkuhpc/analysis/0_preprocess/PBMC/blood_R2.fq.gz \
> /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood.sam
```

SAMtools provide various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format. Here, we use `samtools view` to convert SAM file to BAM file via

`samtools view -bS -@ 4 OC.sam > OC.bam`.

You can specify available header from the input of SAM file (-S), and output in BAM format (-b). The -@ option specifies the number of threads to use.

Let's talk about the reasons of this conversion.

SAM files are a type of text file format that contains the alignment information of various sequences that are mapped against reference sequences. These files can also contain unmapped sequences. Since SAM files are a text file format, **they are more readable by humans**.

BAM files contain the same information as SAM files, except they are in **binary file format** which is not readable by humans. On the other hand, BAM files are **smaller and more efficient** for software to work with than SAM files, **saving time and reducing costs of computation and storage**. Alignment data is almost always stored in BAM files and most software that analyzes aligned reads expects to ingest data in BAM format (often with a BAM index file, to be discussed later in this post).

The remainder of this piece will refer to just the BAM file for simplicity, although the data are identical between SAM and BAM files.

Many of the downstream analysis programs that use BAM files actually require a sorted BAM file. This allows access to reads to be done more efficiently. To sort a BAM file, you can simply use

`samtools sort /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.bam -o /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.bam`

`samtools sort /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood.bam -o /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sort`

You must specify the path and file name of the outputs (-o), and can specify number of threads (-@) and memory (-m) if necessary.

4. Mark duplications

Duplicates are sets of reads pairs that have the same unclipped alignment start and unclipped alignment end. They're sampled from the exact same template of DNA and are suspected to be non-independent measurements of a sequence, which Violates assump7ons of variant calling. To mark duplications, we will use the famous `gatk`.

Genome Analysis Toolkit (GATK) is designed to enable the rapid development of efficient and robust analysis. Its scope is now expanding to include somatic short variant calling, and to tackle copy number (CNV) and structural variation (SV). In addition to the variant callers themselves, the GATK also includes many utilities to perform related tasks such as processing and quality control.

Duplications might come from the same input DNA template, so we can assume that reads will have same start position on the reference. GATK identifies duplicated sets, then chooses representative reads based on base quality scores and other criteria. Pratically, you can mark duplications via

```
gatk MarkDuplicates -I /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.bam \
 -O /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup.bam \
 -M /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup_matrices.txt
```

```
gatk MarkDuplicates -I /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.bam \
 -O /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup.bam \
 -M /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup_matrices.txt
```

You can specify the file to write statistic metrics of duplications by -M option.

5. Base quality score recalibration

As we mentioned before, FASTQ files contains the quality score for filtering and quality evaluation. Base quality scores are per-base estimates of error emitted by the sequencing machines; they express how confident the machine was that it called the correct base each time. For example, let's say the machine reads an A nucleotide, and assigns a quality score of Q20 -- in Phred-scale, that means it's 99% sure it identified the base correctly.

This may seem high, but it does mean that we can expect it to be wrong in one case out of 100; so if we have several billion base calls (we get ~90 billion in a 30x genome), at that rate the machine would make the wrong call in 900 million bases -- which is a lot of bad bases.

Because our short variant calling algorithms rely heavily on the quality score assigned to the individual base calls in each sequence read. This is because the quality score tells us how much we can trust that particular observation to inform us about the biological truth of the site where that base aligns.

Unfortunately the scores produced by the machines are subject to various sources of systematic (non-random) technical error, leading to over- or under-estimated base quality scores in the data. Some of these errors are due to the physics or the chemistry of how the sequencing reaction works, and some are probably due to manufacturing flaws in the equipment.

Base quality score recalibration (BQSR) is a process in which we apply machine learning to model these errors empirically and adjust the quality scores accordingly. For example we can identify that, for a given run, whenever we called two A nucleotides in a row, the next base we called had a 1% higher rate of error. So any base call that comes after AA in a read should have its quality score reduced by 1%. We do that over several different covariates (mainly sequence context and position in read, or cycle) in a way that is additive. So the same base may have its quality score increased for one reason and decreased for another.

So, in order to do BQSR, we can do these two steps:

a. Generates a recalibration table based on various covariates by calculating all the reads that needs recalibration.

```
gatk BaseRecalibrator -R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup.bam \
--known-sites /lustre1/share/references/1000G_phase1.snps.high_confidence.hg38.vcf \
--known-sites /lustre1/share/references/Mills_and_1000G_gold_standard.indels.hg38.vcf \
--known-sites /lustre1/share/references/dbsnp_138.hg38.vcf \
-O /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.recal_data.table
```

```
gatk BaseRecalibrator -R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup.bam \
--known-sites /lustre1/share/references/1000G_phase1.snps.high_confidence.hg38.vcf \
--known-sites /lustre1/share/references/Mills_and_1000G_gold_standard.indels.hg38.vcf \
--known-sites /lustre1/share/references/dbsnp_138.hg38.vcf \
-O /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood.recal_data.table
```

As you can see, we use 3 vcf file from the GATK resource bundle.

1000G*phase1.snps.high*confidence.hg38.vcf --- the 1000 Genomes Phase I indel calls with high confidence of SNP

Mills*and*1000G*gold*standard.indels.b37.sites.vcf --- one of the best sets of known indels to be used for local realignment

dbsnp_138.hg38.vcf --- A recent dbSNP release (build 138) subsetted to only sites discovered in or before dbSNPBuildID 129, which excludes the impact of the 1000 Genomes project and is useful for evaluation of dbSNP rate and Ti/Tv values at novel sites.

b. Apply numerical corrections to each individual basecall based on the patterns identified in **a** (recorded in the recalibration table) and write out the recalibrated data to a new BAM file. The goal of this procedure is to correct for systematic bias that affect the assignment of base quality scores by the sequencer.

```
gatk ApplyBQSR -R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup.bam \
--bqsr-recal-file /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.recal_data.table \
-O /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup.BQSR.bam
```

```
gatk ApplyBQSR -R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup.bam \
--bqsr-recal-file /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.recal_data.table \
-O /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup.BQSR.bam
```

This allows us to get more accurate base qualities overall, which in turn improves the accuracy of our variant calls. To be clear, we can't correct the base calls themselves, i.e. we can't determine whether that low-quality A should actually have been a T -- but we can at least tell the variant caller more accurately how far it can trust that A.

6. Variant discovery

Single-nucleotide polymorphisms (SNPs) are the most common form of genetic variation in humans and drive phenotypic variation. In general, there exist two major types of sequence variations, transition and transversion SNPs, as well as insertions and deletions (indels).

The `gatk HaplotypeCaller` is capable of calling SNPs and indels simultaneously via local de-novo assembly of haplotypes in an active region. In other words, whenever the program encounters a region showing signs of variation, it discards the existing mapping information and completely reassembles the reads in that

region. This allows the HaplotypeCaller to be more accurate when calling regions that are traditionally difficult to call, for example when they contain different types of variants close to each other. It also makes the HaplotypeCaller much better at calling indels than position-based callers.

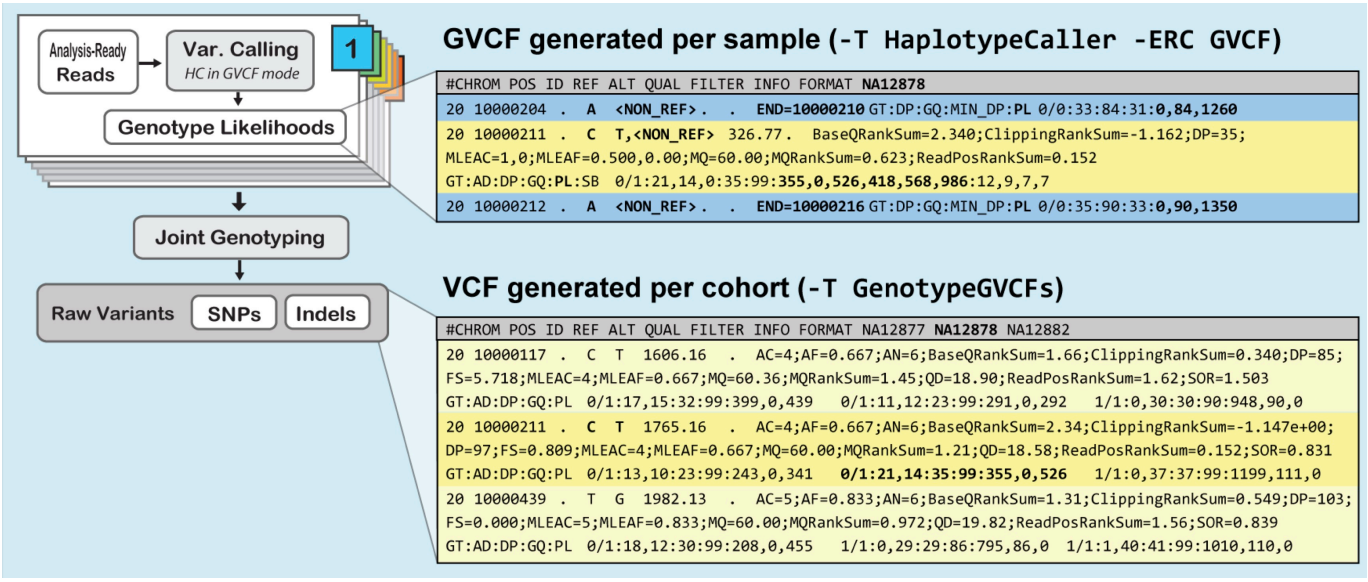
Firstly, HaplotypeCaller defines active regions and determines which regions of the genome it needs to operate on (active regions), based on the presence of evidence for variation. You can use `gatk HaplotypeCaller` via

```
gatk HaplotypeCaller -ERC GVCF \
-R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/OC/OC_sorted.markdup.BQSR.bam \
-D /lustre1/share/references/dbsnp_138.hg38.vcf \
-O /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.g.vcf
```

```
gatk HaplotypeCaller -ERC GVCF \
-R /lustre1/share/references/hg38.fa \
-I /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood_sorted.markdup.BQSR.bam \
-D /lustre1/share/references/dbsnp_138.hg38.vcf \
-O /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood.g.vcf
```

Given that we have run `gatk HaplotypeCaller` for OC and blood sample, respectively, we will get 2 `.g.vcf` file. The `vcf` stands for vairant call format. These files produced for germline short variant (SNP and indel) calls.

GVCF stands for Genomic VCF. The key difference between a regular VCF and a GVCF is that the GVCF has records for all sites, whether there is a variant call there or not. The goal is to have every site represented in the file in order to do joint analysis of a cohort in subsequent steps. The records in a GVCF include an accurate estimation of how confident we are in the determination that the sites are homozygous-reference or not.



After-class Assignments (1)

1. Briefly indroduce the structure and major contens of `.vcf` file? (15')
2. What information can we get from the header? (15')
3. How many fields are required by VCF format and what are they? And how to interpret the variant call records and genotype? (20')

Secondly, it determines haplotypes by assembly of the active region. For each active region, it builds a De Bruijn-like graph to reassemble the active region and identifies what are the possible haplotypes present in the data. It then realigns each haplotype against the reference haplotype using the Smith-Waterman algorithm in order to identify potentially variant sites.

Now, we mkdir a new directory, `mkdir 2_variant_call` for combined OC and blood `VCF` file. We can simply combine both GVCF files via

```
gatk CombineGVCFs -R /lustre1/share/references/hg38.fa \
-V /home/teach31_pkuhpc/analysis/1_alignment/OC/OC.g.vcf \
-V /home/teach31_pkuhpc/analysis/1_alignment/PBMC/blood.g.vcf \
-O /home/teach31_pkuhpc/analysis/2_variant_call/combined.g.vcf
```

Then, HaplotypeCaller determines likelihoods of the haplotypes given the read data. For each active region, it performs a pairwise alignment of each read against each haplotype using the PairHMM algorithm. This produces a matrix of likelihoods of haplotypes given the read data. These likelihoods are then marginalized to obtain the likelihoods of alleles for each potentially variant site given the read data.

Finally, it assigns sample genotypes. For each potentially variant site, it applies Bayes' rule, using the likelihoods of alleles given the read data to calculate the likelihoods of each genotype per sample given the read data observed for that sample. The most likely genotype is then assigned to the sample.

We can perform joint genotyping on a single input, which may contain one or many samples via

```
gatk  GenotypeGVCFs -R /lustre1/share/references/hg38.fa \
-V /home/teach31_pkuhpc/analysis/2_variant_call/combined.g.vcf \
-G StandardAnnotation \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf
```

In this way, we get a final `.vcf` file in which all samples have been jointly genotyped.

7. Variant quality score recalibration

Variant Quality Score Recalibration (VQSR) is a sophisticated filtering technique applied on the variant callset that uses machine learning to model the technical profile of variants in a training set and uses that to filter out probable artifacts from the callset.

Specifically, VQSR calculates a new quality score called the VQSLOD (for variant quality score log-odds) that takes into account various properties of the variant context not captured in the QUAL score. The purpose of this new score is to enable variant filtering in a way that allows analysts to balance sensitivity (trying to discover all the real variants) and specificity (trying to limit the false positives that creep in when filters get too lenient) as finely as possible.

The VQSR method uses machine learning algorithms to learn from each dataset what is the annotation profile of good variants vs. bad variants, and does so in a way that integrates information from multiple dimensions (like, 5 to 8, typically). It takes the overlap of the training/truth resource sets and of your callset. It models the distribution of these variants relative to the annotations you specified, and attempts to group them into clusters. Then it uses the clustering to assign VQSLOD scores to all variants. Variants that are closer to the heart of a cluster will get a higher score than variants that are outliers. If you are interested in more details of algorithms, try to read the original publications. It will help you improve modeling skills.

Now, let's do variant quality recalibration for SNP and Indel, respectively. To begin with, we need to annotate the variants, so that these annotations can be applied in downstream processing.

```
gatk VariantAnnotator \
-R /lustre1/share/references/hg38.fa \
-V /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.anno.vcf \
-A Coverage \
--dbsnp /lustre1/share/references/dbsnp_138.hg38.vcf
```

* SNP quality recalibration

We can get a recalibration table file via

```
gatk VariantRecalibrator -R /lustre1/share/references/hg38.fa \
 -V /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf \
 --resource:hapmap,known=false,training=true,truth=true,prior=15.0 /lustre1/share/references/hapmap_3.3.hg38.vcf \
 --resource:omni,known=false,training=true,truth=false,prior=12.0 /lustre1/share/references/1000G_omni2.5.hg38.vcf \
 --resource:1000G,known=false,training=true,truth=false,prior=10.0 /lustre1/share/references/1000G_phase1.snps.high_confidence.hg
 --resource:dbsnp,known=true,training=false,truth=false,prior=2.0 /lustre1/share/references/dbsnp_138.hg38.vcf \
-an QD -an MQ -an MQRankSum -an ReadPosRankSum -an FS -an SOR \
-mode SNP \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.recal \
--tranches-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.tranches \
--rscript-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.plots.R
```

In this case, we use 4 resource datasets that GATK recommends, which are HapMap, Omni, 1000G and dbSNP. The meaning of the following parameters are as followed:

known - The program only uses known sites for reporting purposes (to indicate whether variants are already known or novel) training - The program builds the Gaussian mixture model using input variants that overlap with these training sites. truth - The program uses these truth sites to determine where to set the cutoff in VQSLOD sensitivity. prior - A prior probability of being correct

And we expect to get these annotations which should used for calculations:

- **QualByDepth (QD)** is the variant confidence (from the QUAL field) divided by the unfiltered depth of non-hom-ref samples. This annotation is intended to normalize the variant quality in order to avoid inflation caused when there is deep coverage.
- **FisherStrand (FS)** is the Phred-scaled probability that there is strand bias at the site. Strand Bias tells us whether the alternate allele was seen more or less often on the forward or reverse strand than the reference allele. When there little to no strand bias at the site, the FS value will be close to 0.
- **StrandOddsRatio (SOR)** is another way to estimate strand bias using a test similar to the symmetric odds ratio test. SOR was created because FS tends to penalize variants that occur at the ends of exons. Reads at the ends of exons tend to only be covered by reads in one direction and FS gives those variants a bad score. SOR will take into account the ratios of reads that cover both alleles.
- **RMSMappingQuality (MQ)** is the root mean square mapping quality over all the reads at the site. Instead of the average mapping quality of the site, this annotation gives the square root of the average of the squares of the mapping qualities at the site. It is meant to include the standard deviation of the mapping qualities. Including the standard deviation allows us to include the variation in the dataset. A low standard deviation means the values are all close to the mean, whereas a high standard deviation means the values are all far from the mean.When the mapping qualities are good at a site, the MQ will be around 60.
- **MappingQualityRankSumTest (MQRankSum)** is the u-based z-approximation from the Rank Sum Test for mapping qualities. It compares the mapping

qualities of the reads supporting the reference allele and the alternate allele. A positive value means the mapping qualities of the reads supporting the alternate allele are higher than those supporting the reference allele; a negative value indicates the mapping qualities of the reference allele are higher than those supporting the alternate allele. A value close to zero is best and indicates little difference between the mapping qualities.

- **ReadPosRankSumTest (ReadPosRankSum)** is the u-based z-approximation from the Rank Sum Test for site position within reads. It compares whether the positions of the reference and alternate alleles are different within the reads. Seeing an allele only near the ends of reads is indicative of error, because that is where sequencers tend to make the most errors. A negative value indicates that the alternate allele is found at the ends of reads more often than the reference allele; a positive value indicates that the reference allele is found at the ends of reads more often than the alternate allele. A value close to zero is best because it indicates there is little difference between the positions of the reference and alternate alleles in the reads.

Now, we can perform VQSR by the recalibration table file:

```
gatk ApplyVQSR  -R /lustre1/share/references/hg38.fa \
-V /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.VQSR.vcf \
--truth-sensitivity-filter-level 99.5 \
--tranches-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.tranches \
--recal-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.recal \
--mode SNP
```

Now, we have assigned a well-calibrated probability to each variant call in a call set. These probabilities can then be used to filter the variants with a greater level of accuracy and flexibility than can typically be achieved by traditional hard-filter (filtering on individual annotation value thresholds). We can get the SNPs above the threshold, which will be labeled as "PASS" via:

```
cat OC_blood.snp.VQSR.vcf | grep "PASS" > OC_blood.snp.filtered.vcf
```

* Indel quality recalibration

Similarly, we get the recalibration table file via

```
gatk VariantRecalibrator -R /lustre1/share/references/hg38.fa \
 -V /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf \
 --resource:mills,known=false,training=true,truth=true,prior=12.0 /lustre1/share/references/Mills_and_1000G_gold_standard.indels.h
 --resource:dbsnp,known=true,training=false,truth=false,prior=2.0 /lustre1/share/references/dbsnp_138.hg38.vcf \
-an QD -an ReadPosRankSum -an FS -an SOR \
-mode INDEL \
--max-gaussians 4 \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.recal \
--tranches-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.tranches \
--rscript-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.plots.R
```

Then, apply VQSR via

```
gatk ApplyVQSR  -R /lustre1/share/references/hg38.fa \
-V /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood_variants.vcf \
-O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.VQSR.vcf \
--truth-sensitivity-filter-level 99.0 \
--tranches-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.tranches \
--recal-file /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.recal \
--mode INDEL
```

* Merge Indel and SNP

Simply use `MergeVcfs` as:

```
gatk MergeVcfs -O /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.all.VQSR.vcf \
-I /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.indel.VQSR.vcf \
-I /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.snps.VQSR.vcf
```

8. Annotation

As we have extracted and filtered variantions, it is important and necessary to utilize prior information to funcationally annotate genetic variants. ANNOVAR is designed to annotate single nucleotide variants (SNVs) and insertions/deletions, such as examining their functional consequence on genes, inferring cytogenetic bands, reporting functional importance scores, finding variants in conserved regions, or identifying variants reported in the 1000 Genomes Project and dbSNP.

All the datasets used in our case are stored at:

`/lustre1/share/annovar/annovar/humandb`

And you can use all the tools of ANNOVAR at:

`/lustre1/share/annovar/annovar` .

We make a new folder in `analysis`:

`mkdir 3_annotation`

Now, we need to convert the merged `.vcf` file to the format that ANNOVAR requires as inputs using Perl:

```
perl /lustre1/share/annovar/annovar/convert2annovar.pl -format vcf4old \
     /home/teach31_pkuhpc/analysis/2_variant_call/OC_blood.all.VQSR.vcf \
     -includeinfo -comment \
     -outfile /home/teach31_pkuhpc/analysis/3_annotation/OC_blood.all.avinput
```

Then we use the outfile as input to perform annotation:

```
perl /lustre1/share/annovar/annovar/table_annovar.pl /home/teach31_pkuhpc/analysis/3_annotation/OC_blood.all.avinput /lustre1/sha
```

After-class Assignments (2)

Refer to the document of GATK and ANNOVAR, explain what each column represents in the output file of `OC_blood_anno.hg38_multianno.txt` and how to interpret it. (50')