Introduction to DNA-Seq analysis

by Mathieu Bourgey, *Ph.D*

Introduction

This workshop will show you how to launch the various steps of a DNA-Seq pipeline in order to call quality variants.

We will be working on a 1000 genome sample, NA12878. You can find the whole raw data on the 1000 genome website: http://www.1000genomes.org/data

NA12878 is the child of the trio while NA12891 and NA12892 are her parents.

Mother	Father	Child
NA12892	NA12891	NA12878

If you finish early, feel free to perform the same steps on the other two individuals: NA12891 & NA12892.

For practical reasons we subsampled the reads from the sample because running the whole dataset would take way too much time and resources. We're going to focus on the reads extracted from a 300 kbp stretch of chromosome 1

Chromosome	Start	End
chr1	17704860	18004860

This analysis for exome sequencing or Whole genome sequencing. In both the cases the analysis is very similar and the main differences are the use of restricted area of search during the different steps.

Original Setup

Acces to the server

Read these directions for information on how to log into the server.

To connect to the server

Port connection depending on the network you are connected you may use one of this to port to connect to the workshop server:

```
-- TCP connections to the port 22 from eduroam
-- TCP connections to the port 20022 from networks other than eduroam
```

linux / Mac user

```
* Open a terminal
* ssh -p <PORT> <USERNAME>@gw.genome.med.kyoto-u.ac.jp
```

Windows

```
* Open your terminal software (TeraTerm or PuTTy)
* create a new connection using the following information
    .  Server : gw.genome.med.kyoto-u.ac.jp
    .  Port :   22 (from eduroam)  20022 (from the network other than eduroam)
    .  USERNAME, passwd :  provided on the first day of the course
```

Software requirements

These are all already installed, but here are the original links.

- BVATools
- SAMTools
- BWA
- Genome Analysis Toolkit
- Picard
- Trimmomatic
- SnpEff

Environment setup

```
#set up
export SOFT_DIR=/usr/local/bin
export WORK_DIR=$HOME/workshop/VariantCalling

export TRIMMOMATIC_JAR=$SOFT_DIR/trimmomatic.jar
export PICARD_JAR=$SOFT_DIR/picard.jar
export GATK_JAR=$SOFT_DIR/GenomeAnalysisTK.jar
export BVATOOLS_JAR=$SOFT_DIR/bvatools-full.jar
export SNPEFF_HOME=/usr/local/src/snpEff/
```

```
export SNPEFF_JAR=$SNPEFF_HOME/snpEff.jar
export REF=$WORK_DIR/reference/


rm -rf $WORK_DIR
mkdir -p $WORK_DIR
cd $WORK_DIR
ln -s /home/mathieu/cleanCopy/* .
```

Data files

The initial structure of your folders should look like this:

```
ROOT
    |-- raw_reads/              # fastqs from the center (down sampled)
        `-- NA12878/            # Child sample directory
        `-- NA12891/            # Father sample directory
        `-- NA12892/            # Mother sample directory
    |-- scripts/                # command lines scripts
    |-- saved_results/          # precomputed final files
```

Cheat sheets

- Unix comand line cheat sheet
- commands file of this module

First data glance

So you've just received an email saying that your data is ready for download from the sequencing center of your choice.

What should you do?

Fastq files

Let's first explore the fastq file.

Try these commands

```
zless -S raw_reads/NA12878/NA12878_CBW_chr1_R1.fastq.gz
```

These are fastq file.

Could you descride the fastq format?

_____

_____

_____

```
zcat raw_reads/NA12878/NA12878_CBW_chr1_R1.fastq.gz | head -n4
zcat raw_reads/NA12878/NA12878_CBW_chr1_R2.fastq.gz | head -n4
```

What was special about the output and why was it like that?

_____

_____

_____

Quality

We can't look at all the reads. Especially when working with whole genome 30x data. You could easilly have Billions of reads.

Tools like FastQC and BVATools readsqc can be used to plot many metrics from these data sets.

Let's look at the data:

```
mkdir -p originalQC/
java -Xmx8G -jar ${BVATOOLS_JAR} readsqc \
  --read1 raw_reads/NA12878/NA12878_CBW_chr1_R1.fastq.gz \
  --read2 raw_reads/NA12878/NA12878_CBW_chr1_R2.fastq.gz \
  --threads 1 --regionName ACTL8 --output originalQC/
```

Copy the images from the originalQC folder to your desktop and open the images following these directions

To copy data from the server

Port connection depending on the network you are connected you may use one of this to port to connect to the workshop server:

```
-- TCP connections to the port 22 from eduroam
-- TCP connections to the port 20022 from networks other than eduroam
```

linux / Mac user

```
* Open a terminal
* scp -r -P <PORT> <USERNAME>@gw.genome.med.kyoto-u.ac.jp:<DISTANT_PATH> <LOCAL_PATH>
```

Windows

```
* Open your file  transfert software (WinSCP or FilleZilla)
* create a new connection using the following information
    .  Server : gw.genome.med.kyoto-u.ac.jp
    .  Port :   22 (from eduroam)  20022 (from the network other than eduroam)
    .  USERNAME, passwd :  provided on the first day of the course
    .  Navigate to the <DISTANT_PATH>
    .  Copy the data into your <LOCAL_PATH>
```

What stands out in the graphs?

All the generated graphics have their uses. This being said 2 of them are particularly useful to get an overal picture of how good or bad a run went.

These are the Quality box plots

and the nucleotide content graphs.

The Box plot shows the quality distribution of your data. The Graph goes > 100 because both ends are appended one after the other.

The quality of a base is computated using the Phread quality score.

The formula outputs an integer that is encoded using an ASCII table.

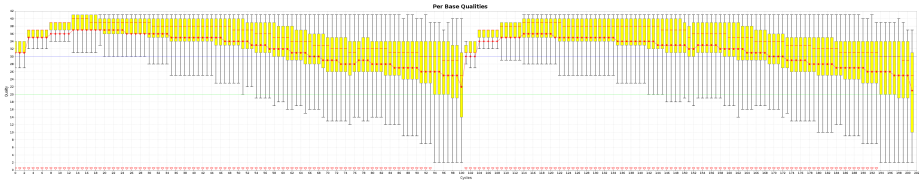

Figure 1: Quality box plots

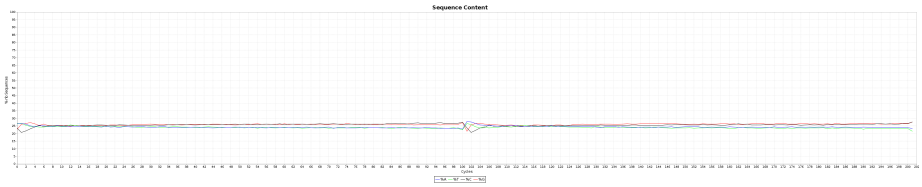

Figure 2: Nucleotide content

The way the lookup is done is by taking the the phred score adding 33 and using this number as a lookup in the table. The Wikipedia entry for the FASTQ format has a summary of the varying values.

Older illumina runs were using phred+64 instead of phred+33 to encode their fastq files.

Trimming

After this careful analysis of the raw data we see that

- Some reads have bad 3' ends.
- no read has adapter sequences in it.

Although nowadays this doesn't happen often, it does still happen. In some cases, miRNA, it is expected to have adapters. Since they are not part of the genome of interest they should be removed if enough reads have them.

To be able to remove adapters and low qualtity beses we will use Trimmomatic.

The adapter file is already in your reference folder.

We can look at the adapters

$$Q_{\mathbf{sanger}} = -10 \log_{10} p$$

Figure 3: Phred quality score formula

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Figure 4: ASCII table

```
cat $REF/adapters.fa
```

Why are there 2 different ones ?

Let's try removing them and see what happens.

```
mkdir -p reads/NA12878/

java -Xmx8G -cp $TRIMMOMATIC_JAR org.usadellab.trimmomatic.TrimmomaticPE -threads 2 -phred33
  raw_reads/NA12878/NA12878_CBW_chr1_R1.fastq.gz \
  raw_reads/NA12878/NA12878_CBW_chr1_R2.fastq.gz \
  reads/NA12878/NA12878_CBW_chr1_R1.t20l32.fastq.gz \
  reads/NA12878/NA12878_CBW_chr1_S1.t20l32.fastq.gz \
  reads/NA12878/NA12878_CBW_chr1_R2.t20l32.fastq.gz \
  reads/NA12878/NA12878_CBW_chr1_S2.t20l32.fastq.gz \
  ILLUMINACLIP:$REF/adapters.fa:2:30:15 TRAILING:20 MINLEN:32 \
  2> reads/NA12878/NA12878.trim.out

cat reads/NA12878/NA12878.trim.out
```

What does Trimmomatic says it did ?

Let's look at the graphs now

```
mkdir -p postTrimQC/
java -Xmx8G -jar ${BVATOOLS_JAR} readsqc \
  --read1 reads/NA12878/NA12878_CBW_chr1_R1.t20l32.fastq.gz \
  --read2 reads/NA12878/NA12878_CBW_chr1_R2.t20l32.fastq.gz \
  --threads 1 --regionName ACTL8 --output postTrimQC/
```

How does it look now?

Could we have done a better job?

Alignment

The raw reads are now cleaned up of artefacts we can align the read to the reference.

In case you have multiple readsets or library you should align them separatly !

Why should this be done separatly?

```
mkdir -p alignment/NA12878/

bwa mem -M -t 2 \
  -R '@RG\tID:NA12878\tSM:NA12878\tLB:NA12878\tPU:runNA12878_1\tCN:Broad Institute\tPL:ILLUM
  ${REF}/hg19.fa\
  reads/NA12878/NA12878_CBW_chr1_R1.t20l32.fastq.gz \
  reads/NA12878/NA12878_CBW_chr1_R2.t20l32.fastq.gz \
  | java -Xmx8G -jar ${PICARD_JAR} SortSam \
  INPUT=/dev/stdin \
  OUTPUT=alignment/NA12878/NA12878.sorted.bam \
  CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT SORT_ORDER=coordinate MAX_RECORDS_IN_RAM=15
```

Why is it important to set Read Group information?

The details of the fields can be found in the SAM/BAM specifications Here For most cases, only the sample name, platform unit and library one are important.

Lane merging (optional)

In case we generate multiple lane of sequencing or mutliple library. It is not practical to keep the data splited and all the reads should be merge into one massive file.

Since we identified the reads in the BAM with read groups, even after the merging, we can still identify the origin of each read.

SAM/BAM

Let's spend some time to explore bam files.

try

```
samtools view alignment/NA12878/NA12878.sorted.bam | head -n4
```

Here you have examples of alignment results. A full description of the flags can be found in the SAM specification

Try using picards explain flag site to understand what is going on with your reads

The flag is the 2nd column.

What do the flags of the first 4 reads mean?

You can use samtools to filter reads as well.

```
# Say you want to count the *un-aligned* reads, you can use
samtools view -c -f4 alignment/NA12878/NA12878.sorted.bam

# Or you want to count the *aligned* reads you, can use
samtools view -c -F4 alignment/NA12878/NA12878.sorted.bam
```

How many reads mapped and unmapped were there?

Another useful bit of information in the SAM is the CIGAR string. It's the 6th column in the file. This column explains how the alignment was achieved.

- M == base aligns *but doesn't have to be a match*. A SNP will have an M even if it disagrees with the reference.
- I == Insertion
- D == Deletion
- S == soft-clips. These are handy to find un removed adapters, viral insertions, etc.

An in depth explanation of the CIGAR can be found here The exact details of the cigar string can be found in the SAM spec as well. Another good site

Cleaning up alignments

We started by cleaning up the raw reads. Now we need to fix and clean some alignments.

Indel realignment (optional if you use GATK HaplotypeCaller variant caller)

The first step for this is to realign around indels and snp dense regions. The Genome Analysis toolkit has a tool for this called IndelRealigner.

It basically runs in 2 steps

1- Find the targets 2- Realign them.

```
java -Xmx8G  -jar ${GATK_JAR} \
  -T RealignerTargetCreator \
  -R ${REF}/hg19.fa\
  -o alignment/NA12878/realign.intervals \
  -I alignment/NA12878/NA12878.sorted.bam \
  -L chr1

java -Xmx8G -jar ${GATK_JAR} \
  -T IndelRealigner \
  -R ${REF}/hg19.fa\
  -targetIntervals alignment/NA12878/realign.intervals \
  -o alignment/NA12878/NA12878.realigned.sorted.bam \
  -I alignment/NA12878/NA12878.sorted.bam
```

How could we make this go faster?

How many regions did it think needed cleaning?

Mark duplicates

As the step says, this is to mark duplicate reads.

What are duplicate reads ?

What are they caused by ?

Here we will use picards for duplicates marking:

```
java -Xmx8G -jar ${PICARD_JAR} MarkDuplicates \
  REMOVE_DUPLICATES=false VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true \
  INPUT=alignment/NA12878/NA12878.realigned.sorted.bam \
  OUTPUT=alignment/NA12878/NA12878.sorted.dup.bam \
  METRICS_FILE=alignment/NA12878/NA12878.sorted.dup.metrics
```

We can look in the metrics output to see what happened.

```
less alignment/NA12878/NA12878.sorted.dup.metrics
```

How many duplicates were there ?

This is very low, we expect in general <2%.

Recalibration

This is the last BAM cleaning-up step.

The goal for this step is to try to recalibrate base quality scores. The vendors tend to inflate the values of the bases in the reads. Also, this step tries to lower the scores of some biased motifs for some technologies.

It runs in 2 steps, 1- Build covariates based on context and known snp sites 2- Correct the reads based on these metrics

```
java -Xmx8G -jar ${GATK_JAR} \
  -T BaseRecalibrator \
  -R ${REF}/hg19.fa\
  -knownSites ${REF}/dbSNP_135_chr1.vcf.gz \
  -L chr1:17700000-18100000 \
  -o alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp \
  -I alignment/NA12878/NA12878.sorted.dup.bam

java -Xmx8G -jar ${GATK_JAR} \
  -T PrintReads \
  -R ${REF}/hg19.fa\
  -BQSR alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp \
  -o alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -I alignment/NA12878/NA12878.sorted.dup.bam
```

Extract Metrics (optional in the workshop)

Once your whole bam is generated, it's always a good thing to check the data again to see if everything makes sens.

Compute coverage (optional in the workshop)

If you have data from a capture kit, you should see how well your targets worked

Both GATK and BVATools have depth of coverage tools. We wrote our own in BVAtools because - GATK was deprecating theirs, but they changed their mind - GATK's is very slow - We were missing some output that we wanted from the GATK's one (GC per interval, valid pairs, etc)

Here we'll use the GATK one

```
java  -Xmx8G -jar ${GATK_JAR} \
  -T DepthOfCoverage \
  --omitDepthOutputAtEachBase \
  --summaryCoverageThreshold 10 \
  --summaryCoverageThreshold 25 \
  --summaryCoverageThreshold 50 \
  --summaryCoverageThreshold 100 \
  --start 1 --stop 500 --nBins 499 -dt NONE \
  -R ${REF}/hg19.fa\
  -o alignment/NA12878/NA12878.sorted.dup.recal.coverage \
  -I alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -L chr1:17700000-18100000

#### Look at the coverage
less -S alignment/NA12878/NA12878.sorted.dup.recal.coverage.sample_interval_summary
```

Coverage is the expected ~30x.

summaryCoverageThreshold is a usefull function to see if your coverage is uniform. Another way is to compare the mean to the median. If both are almost equal, your coverage is pretty flat. If both are quite different, that means something is wrong in your coverage. A mix of WGS and WES would show very different mean and median values.

Insert Size (optional in the workshop)

```
java -Xmx8G -jar ${PICARD_JAR} CollectInsertSizeMetrics \
  VALIDATION_STRINGENCY=SILENT \
  REFERENCE_SEQUENCE=${REF}/hg19.fa\
  INPUT=alignment/NA12878/NA12878.sorted.dup.recal.bam \
  OUTPUT=alignment/NA12878/NA12878.sorted.dup.recal.metric.insertSize.tsv \
  HISTOGRAM_FILE=alignment/NA12878/NA12878.sorted.dup.recal.metric.insertSize.histo.pdf \
  METRIC_ACCUMULATION_LEVEL=LIBRARY

#look at the output
less -S alignment/NA12878/NA12878.sorted.dup.recal.metric.insertSize.tsv
```

What is the insert size and the corresponding standard deviation ?

Is the insert-size important ?

Alignment metrics (optional in the workshop)

For the alignment metrics, we used to use `samtools flagstat` but with bwa mem since some reads get broken into pieces, the numbers are a bit confusing. You can try it if you want.

We prefer the Picard way of computing metrics

```
java -Xmx8G -jar ${PICARD_JAR} CollectAlignmentSummaryMetrics  \
  VALIDATION_STRINGENCY=SILENT \
  REFERENCE_SEQUENCE=${REF}/hg19.fa\
  INPUT=alignment/NA12878/NA12878.sorted.dup.recal.bam \
  OUTPUT=alignment/NA12878/NA12878.sorted.dup.recal.metric.alignment.tsv \
  METRIC_ACCUMULATION_LEVEL=LIBRARY

#### explore the results

less -S alignment/NA12878/NA12878.sorted.dup.recal.metric.alignment.tsv
```

What is the percent of aligned reads ?

_____

_____

_____

Calling variants with GATK

We mapped the reads to hg19 and then we removed duplicate reads and realigned the reads around the indels. Let's now call SNPs in NA12878 :

```
mkdir -p variants/
java -Xmx8g -jar $GATK_JAR -T HaplotypeCaller -l INFO -R ${REF}/hg19.fa\
-I alignment/NA12878/NA12878.sorted.dup.recal.bam  --variant_index_type LINEAR --variant_ind
-o variants/NA12878.hc.vcf -L chr1:17704860-18004860
```

`-Xmx8g` instructs java to allow up 2 GB of RAM to be used for GATK.

`-l` INFO specifies the minimum level of logging.

`-R` specifies which reference sequence to use.

`-I` specifies the input BAM files.

`--variant_index_type` LINEAR specifies the indexing strategy to use for VCFs. LINEAR creates a Linear index with bins of equal width, specified by the Bin Width parameter.

`--variant_index_parameter` 128000 specifies the bin width.

`-dt` NONE specifies to do not downsample the data.

`-L` indicates the reference region where SNP calling should take place

HERE WE ARE !! variant have been called

Do you think this is enough ?

Filter the variants

Typically variant callers will only perform a minimal amount of filtering when presenting variant calls.

To perform more rigorous filtering, another program must be used. In our case, we will use the *VariantFiltration* tool in GATK.

NOTE: The best practice when using GATK is to use the *VariantRecalibrator*. In our data set, we had too few variants to accurately use the variant recalibrator and therefore we used the *VariantFiltration* tool instead.

```
java -Xmx8g -jar $GATK_JAR -T VariantFiltration \
-R ${REF}/hg19.fa--variant variants/NA12878.hc.vcf -o variants/NA12878.hc.filter.vcf --filte
--filterExpression "FS > 200.0" \
--filterExpression "MQ < 40.0" \
--filterName QDFilter \
--filterName FSFilter \
--filterName MQFilter
```

`-Xmx8g` instructs java to allow up 8 GB of RAM to be used for GATK.

`-R` specifies which reference sequence to use.

`--variant` specifies the input vcf file.

`-o` specifies the output vcf file.

`--filterExpression` defines an expression using the vcf INFO and genotype variables.

`--filterName` defines what the filter field should display if that filter is true.

What is QD, FS, and MQ?

Adding functional consequence

The next step in trying to make sense of the variant calls is to assign functional consequence to each variant.

At the most basic level, this involves using gene annotations to determine if variants are sense, missense, or nonsense.

We typically use SnpEff but many use Annovar and VEP as well.

Let's run snpEff

```
java -Xmx8G -jar $SNPEFF_JAR eff  -v -no-intergenic \
-i vcf -o vcf hg19 variants/NA12878.hc.filter.vcf >  variants/NA12878.hc.filter.snpeff.vcf
```

`-Xmx8g` instructs java to allow up 4 GB of RAM to be used for snpEff.

`-c` specifies the path to the snpEff configuration file

`-v` specifies verbose output.

`-no-intergenic` specifies that we want to skip functional consequence testing in intergenic regions.

`-i` and `-o` specify the input and output file format respectively. In this case, we specify vcf for both.

`hg19` specifies that we want to use the hg19 annotation database.

`variants/NA12878.hc.filter.vcf` specifies our input vcf filename

`variants/NA12878..hc.filter.snpeff.vcf` specifies our output vcf filename

Investigating the functional consequence of variants

You can learn more about the meaning of snpEff annotations here.

Use less to look at the new vcf file:

```
less -S variants/NA12878.hc.filter.snpeff.vcf
```

We can see in the vcf that snpEff added few sections. These are hard to decipher directly from the VCF other tools or scripts, need to be used to make sens of this.

The annotation is presented in the INFO field using the new ANN format. For more information on this field see here. Typically, we have:

```
ANN=Allele|Annotation|Putative impact|Gene name|Gene ID|Feature
type|Feature ID|Transcript biotype|Rank Total|HGVS.c|...
```

Here's an example of a typical annotation:

```
ANN=C|intron_variant|MODIFIER|PADI6|PADI6|transcript|NM_207421.4|Coding|5/16|c.553+80T>C||||
```

What does the example annotation actually mean?

additionally, you could download and look at the report generated by snpEff `snpEff_summary.html`.

How many variants had a high impact?

How many variants had a moderate impact?

Adding dbSNP annotations

Go back to look at your last vcf file:

```
less -S variants/NA12878.hc.filter.snpeff.vcf
```

What do you see in the third column?

The third column in the vcf file is reserved for identifiers. Perhaps the most common identifier is the dbSNP rsID.

Use the following command to generate dbSNP rsIDs for our vcf file:

```
java -Xmx8g -jar $GATK_JAR -T VariantAnnotator -R ${REF}/hg19.fa \
--dbsnp $REF/dbSNP_135_chr1.vcf.gz --variant variants/NA12878.hc.filter.snpeff.vcf \
-o variants/NA12878.hc.filter.snpeff.dbsnp.vcf -L chr1:17704860-18004860
```

`-Xmx2g` instructs java to allow up 2 GB of RAM to be used for GATK.

`-R` specifies which reference sequence to use.

`--dbsnp` specifies the input dbSNP vcf file. This is used as the source for the annotations.

`--variant` specifies the input vcf file.

`-o` specifies the output vcf file.

`-L` defines which regions we should annotate. In this case, I chose the chromosomes that contain the regions we are investigating.

Can you find a variant that passed and wasn't in dbSNP?

(Optional 1) Use IGV to investigate the SNPs

The best way to see and understand the differences between the two vcf files will be to look at them in IGV.

If you need, the IGV color codes can be found here: IGV color code by insert size and IGV color code by pair orientation.

You can download all the NA12878.* files in the current directory to your local computer:

To do this you can use the procedure that was described previously.

After that you need to follow the steps as in Option 1 except that you need to load the files in IGV using (File->Load from File...).

Finally, go to a region on chromsome 1 with reads (chr1:17704860-18004860) and spend some time SNP gazing...

Do the SNPs look believable?

_____

_____

_____

Are there any positions that you think should have been called as a SNP, but weren't?

_____

_____

_____

(Optional 2) Investigating the trio

At this point we have aligned and called variants in one individual. However, we actually have FASTQ and BAM files for three family members!

As additional practice, perform the same steps for the other two individuals (her parents): NA12891 and NA12892. Here are some additional things that you might want to look at:

1. **If you load up all three realigned BAM files and all three final vcf files into IGV, do the variants look plausible?** Use a Punnett square to help evaluate this. i.e. if both parents have a homozygous reference call and the child has a homozygous variant call at that locus, this might indicate a trio conflict.

2. **Do you find any additional high or moderate impact variants in either of the parents?**

3. **Do all three family members have the same genotype for Rs7538876 and Rs2254135?**

4. GATK produces even better variant calling results if all three BAM files are specified at the same time (i.e. specifying multiple `-I filename` options). Try this and then perform the rest of module 5 on the trio vcf file. **Does this seem to improve your variant calling results? Does it seem to reduce the trio conflict rate?**

Summary

In this lab, we aligned reads from the sample NA12878 to the reference genome `hg19`:

- We became familiar with FASTQ and SAM/BAM formats.
- We checked read QC with BVAtools.
- We trimmed unreliable bases from the read ends using Trimmomatic.
- We aligned the reads to the reference using BWA.
- We sorted the alignments by chromosome position using PICARD.
- We realigned short indels using GATK.
- We recalibrate the Base Quality using GATK.
- We call and filter variants using GATK
- We annotate variant with SnpEff

Acknowledgments

I would like to thank and acknowledge Louis Letourneau, Michael Stromberg and Guillaume Bourque for this help and for sharing his material. The format of the tutorial has been inspired from Mar Gonzalez Porta of Embl-EBI.
