DNA-Seq processing - Kyoto Course on Bioinformatics 2014

Louis Letourneau

2014/03/10

Contents

Introduction to DNA-Seq processing

This workshop will show you how to launch individual steps of a complete DNA-Seq pipeline

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

For practical reasons we subsampled the reads from the sample because running the whole dataset would take way too much time and resources.

Original Setup

The initial structure of your folders should look like this:

```
<ROOT>
|-- raw_reads/               # fastqs from the center (down sampled)
    `-- NA12878              # One sample directory
        |-- runERR_1         # Lane directory by run number. Contains the fastqs
        `-- runSRR_1         # Lane directory by run number. Contains the fastqs
`-- project.nanuq.csv        # sample sheet
```

Cheat sheets

- Unix comand line cheat sheet[2]

[1] http://creativecommons.org/licenses/by-sa/3.0/deed.en_US
[2] http://sites.tufts.edu/cbi/files/2013/01/linux_cheat_sheet.pdf

Environment setup

```
export PATH=$PATH:/home/Louis/tools/tabix-0.2.6/:/home/Louis/tools/igvtools_2.3.31/
export PICARD_HOME=/usr/local/bin
export SNPEFF_HOME=/home/Louis/tools/snpEff_v3_5_core/snpEff
export GATK_JAR=/usr/local/bin/GenomeAnalysisTK.jar
export BVATOOLS_JAR=/home/Louis/tools/bvatools-1.1/bvatools-1.1-full.jar
export TRIMMOMATIC_JAR=/usr/local/bin/trimmomatic-0.32.jar
export REF=/home/Louis/kyotoWorkshop/references/

cd $HOME
rsync -avP /home/Louis/cleanCopy/ $HOME/workshop/
cd $HOME/workshop/
```

Software requirements

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

- FastQC[3]
- BVATools[4]
- SAMTools[5]
- IGV[6]
- BWA[7]
- Genome Analysis Toolkit[8]
- Picard[9]
- SnpEff[10]

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.
The first thing to do is download it, the second thing is making sure it is of good quality.

[3] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
[4] https://bitbucket.org/mugqic/bvatools/downloads
[5] http://sourceforge.net/projects/samtools/
[6] http://www.broadinstitute.org/software/igv/download
[7] http://bio-bwa.sourceforge.net/
[8] http://www.broadinstitute.org/gatk/
[9] http://picard.sourceforge.net/
[10] http://snpeff.sourceforge.net/

Fastq files

Let's first explore the fastq file.

Try these commands

```
zless -S raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz

zcat raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz | head -n4
zcat raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair2.fastq.gz | head -n4
```

From the second set of commands (the head), what was special about the output?
Why was it like that?
Solution[11]

You could also just count the reads

```
zgrep -c "^@SRR" raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz
```

Why shouldn't you just do

```
zgrep -c "^@" raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz
```

Solution[12]

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 originalQC/
java -Xmx1G -jar ${BVATOOLS_JAR} readsqc \
  --read1 raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz \
  --read2 raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair2.fastq.gz \
  --threads 2 --regionName SRR --output originalQC/

java -Xmx1G -jar ${BVATOOLS_JAR} readsqc \
  --read1 raw_reads/NA12878/runERR_1/NA12878.ERR.33.pair1.fastq.gz \
  --read2 raw_reads/NA12878/runERR_1/NA12878.ERR.33.pair2.fastq.gz \
  --threads 2 --regionName ERR --output originalQC/
```

Copy the images from the `originalQC` folder to your desktop and open the images.

```
scp -r <USER>@www.genome.med.kyoto-u.ac.jp:~/workshop/originalQC/ ./
```

[11] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_fastq.ex1.md
[12] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/_fastq.ex2.md

What stands out in the graphs?
Solution[13]

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. In this case the reasons there are spikes and jumps in quality and length is because there are actually different libraries pooled together in the 2 fastq files. The sequencing lengths vary between 36,50,76 bp read lengths. 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.

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

The formula outputs an integer that is encoded using an ASCII[14] table. 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[15] has a summary of the varying values.

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

In the SRR dataset we also see some adapters.
Why does this happen Solution[16]

Trimming

After this careful analysis of the raw data we see that

- Some reads have bad 3' ends.
- Some reads have adapter sequences in them.

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 the adapters we need to feed them to a tool. In this case we will use Trimmomatic. The dapter file is already in your work folder.
We can look at the adapters

```
cat adapters.fa
```

Why are there 2 different ones? Solution[17]

Let's try removing them and see what happens.

[13] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_fastqQC.ex1.md
[14] http://en.wikipedia.org/wiki/ASCII
[15] http://en.wikipedia.org/wiki/FASTQ_format
[16] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_fastqQC.ex2.md
[17] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_trim.ex1.md

```
mkdir -p reads/NA12878/runSRR_1/
mkdir -p reads/NA12878/runERR_1/

java -Xmx2G -cp $TRIMMOMATIC_JAR org.usadellab.trimmomatic.TrimmomaticPE -threads 2 -phred33 \
  raw_reads/NA12878/runERR_1/NA12878.ERR.33.pair1.fastq.gz \
  raw_reads/NA12878/runERR_1/NA12878.ERR.33.pair2.fastq.gz \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair1.fastq.gz \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.single1.fastq.gz \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair2.fastq.gz \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.single2.fastq.gz \
  ILLUMINACLIP:adapters.fa:2:30:15 TRAILING:20 MINLEN:32 \
  2> reads/NA12878/runERR_1/NA12878.ERR.trim.out

java -Xmx2G -cp $TRIMMOMATIC_JAR org.usadellab.trimmomatic.TrimmomaticPE -threads 2 -phred33 \
  raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair1.fastq.gz \
  raw_reads/NA12878/runSRR_1/NA12878.SRR.33.pair2.fastq.gz \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair1.fastq.gz \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.single1.fastq.gz \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair2.fastq.gz \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.single2.fastq.gz \
  ILLUMINACLIP:adapters.fa:2:30:15 TRAILING:20 MINLEN:32 \
  2> reads/NA12878/runSRR_1/NA12878.SRR.trim.out

cat reads/NA12878/runERR_1/NA12878.ERR.trim.out reads/NA12878/runSRR_1/NA12878.SRR.trim.out
```

What does Trimmomatic says it did? Solution[18]

Let's look at the graphs now

```
mkdir postTrimQC/
java -Xmx1G -jar ${BVATOOLS_JAR} readsqc \
  --read1 reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair1.fastq.gz \
  --read2 reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair2.fastq.gz \
  --threads 2 --regionName ERR --output postTrimQC/
java -Xmx1G -jar ${BVATOOLS_JAR} readsqc \
  --read1 reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair1.fastq.gz \
  --read2 reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair2.fastq.gz \
  --threads 2 --regionName SRR --output postTrimQC/
```

How does it look now? Solution[19]

Alignment

The raw reads are now cleaned up of artefacts we can align each lane separatly.

Why should this be done separatly? Solution[20]

[18] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_trim.ex2.md
[19] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_trim.ex3.md
[20] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_aln.ex1.md

```
mkdir -p alignment/NA12878/runERR_1
mkdir -p alignment/NA12878/runSRR_1

bwa mem -M -t 2 \
  -R '@RG\tID:ERR_ERR_1\tSM:NA12878\tLB:ERR\tPU:runERR_1\tCN:Broad Institute\tPL:ILLUMINA' \
  ${REF}/b37.fasta \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair1.fastq.gz \
  reads/NA12878/runERR_1/NA12878.ERR.t20l32.pair2.fastq.gz \
  | java -Xmx2G -jar ${PICARD_HOME}/SortSam.jar \
  INPUT=/dev/stdin \
  OUTPUT=alignment/NA12878/runERR_1/NA12878.ERR.sorted.bam \
  CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT SORT_ORDER=coordinate MAX_RECORDS_IN_RAM=500000

bwa mem -M -t 2 \
  -R '@RG\tID:SRR_SRR_1\tSM:NA12878\tLB:SRR\tPU:runSRR_1\tCN:Broad Institute\tPL:ILLUMINA' \
  ${REF}/b37.fasta \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair1.fastq.gz \
  reads/NA12878/runSRR_1/NA12878.SRR.t20l32.pair2.fastq.gz \
  | java -Xmx2G -jar ${PICARD_HOME}/SortSam.jar \
  INPUT=/dev/stdin \
  OUTPUT=alignment/NA12878/runSRR_1/NA12878.SRR.sorted.bam \
  CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT SORT_ORDER=coordinate MAX_RECORDS_IN_RAM=500000
```

Why is it important to set Read Group information? Solution[21]

The details of the fields can be found in the SAM/BAM specifications Here[22]
For most cases, only the sample name, platform unit and library one are important.

Why did we pipe the output of one to the other? Could we have done it differently? Solution[23]

We will explore the generated BAM latter.

Lane merging

We now have alignments for each of the sequences lanes. This is not practical in it's current form. What we wan't to do now
is merge the results into one BAM.

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

```
java -Xmx2G -jar ${PICARD_HOME}/MergeSamFiles.jar \
  INPUT=alignment/NA12878/runERR_1/NA12878.ERR.sorted.bam \
  INPUT=alignment/NA12878/runSRR_1/NA12878.SRR.sorted.bam \
  OUTPUT=alignment/NA12878/NA12878.sorted.bam \
  VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true
```

You should now have one BAM containing all your data.
Let's double check

[21] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_aln.ex2.md
[22] http://samtools.sourceforge.net/SAM1.pdf
[23] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_aln.ex3.md

```
ls -l alignment/NA12878/
samtools view -H alignment/NA12878/NA12878.sorted.bam | grep "^@RG"
```

You should have your 2 read group entries.
Why did we use the `-H` switch? Try without. What happens? Solution[24]

SAM/BAM

Let's spend some time to explore bam files.

try

```
samtools view alignment/NA12878/NA12878.sorted.bam | head -n2
```

Here you have examples of alignment results.
A full description of the flags can be found in the SAM specification
http://samtools.sourceforge.net/SAM1.pdf

Try using picards explain flag site to understand what is going on with your reads
http://picard.sourceforge.net/explain-flags.html

The flag is the 2nd column.

What do the flags of the first 2 reads mean? Solution[25]

Let's take the 2nd one, the one that is in proper pair, and find it's pair.

try

```
samtools view alignment/NA12878/NA12878.sorted.bam | grep ERR001733.2317763
```

Why did searching one name find both reads? Solution[26]

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? Solution[27]

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.

[24] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_merge.ex1.md
[25] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_sambam.ex1.md
[26] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_sambam.ex2.md
[27] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_sambam.ex3.md

An in depth explanation of the CIGAR can be found here[28]
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 some alignments.

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 -Xmx2G  -jar ${GATK_JAR} \
  -T RealignerTargetCreator \
  -R ${REF}/b37.fasta \
  -o alignment/NA12878/realign.intervals \
  -I alignment/NA12878/NA12878.sorted.bam \
  -L 1

java -Xmx2G -jar ${GATK_JAR} \
  -T IndelRealigner \
  -R ${REF}/b37.fasta \
  -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? Solution[29]
How many regions did it think needed cleaning? Solution[30]

Indel Realigner also makes sure the called deletions are left aligned when there is a microsat of homopolmer.

```
This
ATCGAAAA-TCG
into
ATCG-AAAATCG

or
ATCGATATATATA--TCG
into
ATCG--ATATATATATCG
```

This makes it easier for down stream tools.

[28] http://genome.sph.umich.edu/wiki/SAM
[29] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_realign.ex1.md
[30] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_realign.ex2.md

FixMates

This step shouldn't be necessary... but it is.

This goes through the BAM file and find entries which don't have their mate information written properly.

This used to be a problem in the GATKs realigner, but they fixed it. It shouldn't be a problem with aligners like BWA, but there are always corner cases that create
one-off corrdinates and such.

This happened a lot with bwa backtrack. This happens less with bwa mem, but it still happens none the less.

```
java -Xmx2G -jar ${PICARD_HOME}/FixMateInformation.jar \
  VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true SORT_ORDER=coordinate MAX_RECORDS_IN_RAM=500000 \
  INPUT=alignment/NA12878/NA12878.realigned.sorted.bam \
  OUTPUT=alignment/NA12878/NA12878.matefixed.sorted.bam
```

Mark duplicates

As the step says, this is to mark duplicate reads.
What are duplicate reads? What are they caused by? Solution[31]
What are the ways to detect them? Solution[32]

Here we will use picards approach:

```
java -Xmx2G -jar ${PICARD_HOME}/MarkDuplicates.jar \
  REMOVE_DUPLICATES=false CREATE_MD5_FILE=true VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true \
  INPUT=alignment/NA12878/NA12878.matefixed.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.
We can see that it computed seperate measures for each library.
Why is this important to do and not combine everything? Solution[33]

How many duplicates were there? Solution[34]

This is on the high side, usually or rather, now since this is old data, this should be <2% for 2-3 lanes.

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.

[31] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_markdup.ex1.md
[32] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_markdup.ex2.md
[33] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_markdup.ex3.md
[34] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_markdup.ex4.md

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

```
java -Xmx2G -jar ${GATK_JAR} \
  -T BaseRecalibrator \
  -nct 2 \
  -R ${REF}/b37.fasta \
  -knownSites ${REF}/dbSnp-137.vcf.gz \
  -L 1:47000000-47171000 \
  -o alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp \
  -I alignment/NA12878/NA12878.sorted.dup.bam

java -Xmx2G -jar ${GATK_JAR} \
  -T PrintReads \
  -nct 2 \
  -R ${REF}/b37.fasta \
  -BQSR alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp \
  -o alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -I alignment/NA12878/NA12878.sorted.dup.bam
```

Just to see how things change let's make GATK recalibrate after a first pass

```
java -Xmx2G -jar ${GATK_JAR} \
  -T BaseRecalibrator \
  -nct 2 \
  -R ${REF}/b37.fasta \
  -knownSites ${REF}/dbSnp-137.vcf.gz \
  -L 1:47000000-47171000 \
  -o alignment/NA12878/NA12878.sorted.dup.recalibration_report.seconnd.grp \
  -I alignment/NA12878/NA12878.sorted.dup.bam \
  -BQSR alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp

java -Xmx2G -jar ${GATK_JAR} \
  -T AnalyzeCovariates \
  -R ${REF}/b37.fasta \
  -before alignment/NA12878/NA12878.sorted.dup.recalibration_report.grp \
  -after alignment/NA12878/NA12878.sorted.dup.recalibration_report.seconnd.grp \
  -csv BQSR.csv \
  -plots BQSR.pdf
```

The graphs don't mean much because we downsampled the data quite a bit. With a true whole genome or whole exome dataset we can see a bigger effect.

Extract Metrics

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

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 come output that we wanted from the GATK's one (GC per interval, valid pairs, etc)

Here we'll use the GATK one

```
java  -Xmx2G -jar ${GATK_JAR} \
  -T DepthOfCoverage \
  --omitDepthOutputAtEachBase \
  --summaryCoverageThreshold 10 \
  --summaryCoverageThreshold 25 \
  --summaryCoverageThreshold 50 \
  --summaryCoverageThreshold 100 \
  --start 1 --stop 500 --nBins 499 -dt NONE \
  -R ${REF}/b37.fasta \
  -o alignment/NA12878/NA12878.sorted.dup.recal.coverage \
  -I alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -L 1:47000000-47171000

# 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

```
java -Xmx2G -jar ${PICARD_HOME}/CollectInsertSizeMetrics.jar \
  VALIDATION_STRINGENCY=SILENT \
  REFERENCE_SEQUENCE=${REF}/b37.fasta \
  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
```

There is something interesting going on with our library ERR.
From the pdf or the tab seperated file, can you tell what it is? Solution[35]

Alignment metrics

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 -Xmx2G -jar ${PICARD_HOME}/CollectAlignmentSummaryMetrics.jar \
  VALIDATION_STRINGENCY=SILENT \
  REFERENCE_SEQUENCE=${REF}/b37.fasta \
  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
```

Variant calling

Here we will try 3 variant callers.
I won't go into the details of finding which variant is good or bad since this will be your next workshop.
Here we will just call and view the variants.

Start with:

```
mkdir variants
```

Samtools

```
samtools mpileup -L 1000 -B -q 1 -D -S -g \
  -f ${REF}/b37.fasta \
  -r 1:47000000-47171000 \
  alignment/NA12878/NA12878.sorted.dup.recal.bam \
  | bcftools view -vcg - \
  > variants/mpileup.vcf
```

GATK Unified Genotyper

```
java -Xmx2G -jar ${GATK_JAR} \
  -T UnifiedGenotyper \
  -R ${REF}/b37.fasta \
  -I alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -o variants/ug.vcf \
```

[35] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_insert.ex1.md

```
  --genotype_likelihoods_model BOTH \
  -dt none \
  -L 1:46000000-47600000
```

GATK Haplotyper

```
java -Xmx2G -jar ${GATK_JAR} \
  -T HaplotypeCaller \
  -R ${REF}/b37.fasta \
  -I alignment/NA12878/NA12878.sorted.dup.recal.bam \
  -o variants/haplo.vcf \
  -dt none \
  -L 1:46000000-47600000
```

Now we have variants from all three methods. Let's compress and index the vcfs for futur visualisation.

```
for i in variants/*.vcf;do bgzip -c $i > $i.gz ; tabix -p vcf $i.gz;done
```

Let's look at a compressed vcf.

```
zless -S variants/mpileup.vcf.gz
```

Details on the spec can be found here:
http://vcftools.sourceforge.net/specs.html

Fields vary from caller to caller. Some values are more constant.
The ref vs alt alleles, variant quality (QUAL column) and the per-sample genotype (GT) values are almost always there.

Annotations

We typically use snpEff but many use annovar and VEP as well.

Let's run snpEff

```
java  -Xmx6G -jar ${SNPEFF_HOME}/snpEff.jar \
  eff -v -c ${SNPEFF_HOME}/snpEff.config \
  -o vcf \
  -i vcf \
  -stats variants/mpileup.snpeff.vcf.stats.html \
  GRCh37.74 \
  variants/mpileup.vcf \
  > variants/mpileup.snpeff.vcf
```

```
less -S variants/mpileup.snpeff.vcf
```

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

For now we will skip this step since you will be working with gene annotations in your next workshop.

Take a look at the HTML stats file snpEff created. It contains some metrics on the variants it analysed.

Visualisation

Before jumping into IGV, we'll generate a track IGV can use to plot coverage.

Try this:

```
igvtools count \
  -f min,max,mean \
  alignment/NA12878/NA12878.sorted.dup.recal.bam \
  alignment/NA12878/NA12878.sorted.dup.recal.bam.tdf \
  b37
```

IGV

You can get IGV here[36]

Open it and choose b37 as the genome

Open your BAM file, the tdf we just generated should load.
Load your vcfs as well.

Find an indel. What's different between the snp callers? Solution[37]
Go to 1:47050562-47051207 what is interesting here? Solution[38]

Look around...

Aknowledgments

The format for this tutorial has been inspired from Mar Gonzalez Porta of Embl-EBI, who I would like to thank and acknowledge. I also want to acknowledge Mathieu Bourgey, Francois Lefebvre, Maxime Caron and Guillaume Bourque for the help in building these pipelines and working with all the various datasets.

[36] http://www.broadinstitute.org/software/igv/download
[37] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_vis.ex1.md
[38] https://github.com/lletourn/Workshops/blob/kyoto201403/blob/solutions/_vis.ex2.md