solution_data

The first thing to do is download it

The second thing is making sure it is of good quality.

solution_fastq1

The 4 lines of each read contain:

```
- Header 1
- DNA sequence
- Header 2
- Quality values
```

solution_fastq2

It's the same header with a /1 or /2 towards the end. Meaning, it's paired data.

solution_fastq3

Because the ASCII quality character has @ as a valid value. If the quality line starts with this character you'll count it as a read

By this method still 4003 counts are found due to the use of the old phred-64 format (which excluded the @ value) but recent data will contain it

solution_fastqQC1

Of the raw data we see that:

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

solution_adapter1

Because the sequenced molecule was shorted than the number of cycles used.

solution_trim1

Because both ends of the fragment don't have the same adapter.

solution_trim2

Of the 4003 input pairs 94.6% were kept 2.8% had only a valid read1 2.3% had only a valid read2 0.3% were fully discarded

solution_fastqQC2

Here is the command to generate the new quality graphs:

```
# Generate post-trimmed QC
mkdir postTrimQC/
java -Xmx1G -jar ${BVATOOLS_JAR} readsqc --quality 33 \
  --read1 reads/normal/runD0YR4ACXX_1/normal.t30l50.pair1.fastq.gz \
  --read2 reads/normal/runD0YR4ACXX_1/normal.t30l50.pair2.fastq.gz \
  --threads 2 --regionName normalD0YR4ACXX_1 --output postTrimQC/
```

solution_trim3

It looks better, and most of adapters have been removed.

solution_aln1

For speed, you can align each in parallel

To track where the reads came from. We will set individual Read Group tags

solution_aln3

Mostly to save IO operations and space. Piping skips the SAM generation

The problem though is that more RAM and processors are needed since we sort the output automatically

solution_aln4

One option is to just generate an unsorted BAM piping bwa-mem in samtools

Another option is to do it in 2 steps and pay the space cost.

solution_merge1

-H is used to only output the header of the BAM/SAM file.

solution_merge2

If it's not given the default is to skip the header and only show alignments.

solution_sambam1

Flag 129:

```
The read is paired
The read is the second in the pair
The read is on forward strand
```

Flag 145:

```
The read is paired
The read is the second in the pair
The read is on reverse strand
```

Flag 419:

```
The read is paired
The read is mapped in proper pair (correct insert size and correct orientation)
The read is the second in the pair
The mate is on reverse strand
The read is not  the primary alignment (best alignement somewhere else)
```

Flag 81:

```
The read is paired
The read is the first in the pair
The mate is on reverse strand
```

By looking at the read names you can notice that these 4 entries represent 4 read of 4 different pairs

solution_sambam3

to catch the read from the pair:

```
samtools view alignment/normal/normal.sorted.bam |grep "HISEQ9_0205:4:1207:20646:145063"
```

solution_sambam4

The first read is aligneed at 2 different locations. None of the shows a perfect alignment which could be considered the primary alignment.

the best alignement is found because of the presence of the mate and his location.

solution_sambam2

If you want to count the *un-aligned* reads you can use:

```
samtools view -c -f4 alignment/normal/normal.sorted.bam
```

Number of unmapped reads :

```
25
```

Or if you want to count the *aligned* reads you can use:

```
samtools view -c -F4 alignment/normal/normal.sorted.bam
```

Number of mapped reads :

```
204626
```

solution_realign3

Because if a region needs realignment, maybe one of the sample in the pair has less reads or was excluded from the target creation.

This makes sur the normal and tumor are all in-sync for the somatic calling step.

solution_realign1

we can break up the search and realignment by chromosome

solution_realign2

```
wc -l alignment/normal/realign.intervals
```

1

solution_realign4

This makes it easier for down stream analysis tools

For NGS analysis, the convention is to left align indels.

This is only really needed when calling variants with legacy locus-based tools such as samtools or GATK UnifiedGenotyper. Otherwise you will have worse performance and accuracy.

With more sophisticated tools (like GATK HaplotypeCaller) that involve reconstructing haplotypes (eg through reassembly), the problem of multiple valid representations is handled internally and does not need to be corrected explicitly.

solution_markdup1

Different read pairs representing the same initial DNA fragment.

solution_markdup2

PCR reactions (PCR duplicates) Some clusters that are thought of being separate in the flowcell but are the same (optical duplicates)

solution_markdup3

Picard and samtools uses the alignment positions:

- Both 5' ends of both reads need to have the same positions.
- Each reads have to be on the same strand as well.

Another method is to use a kmer approach:

- take a part of both ends of the fragment
- build a hash table
- count the similar hits

Brute force, compare all the sequences.

solution_markdup4

- SRR 3%
- ERR 7%

solution_markdup5

Each library represents a set of different DNA fragments.

Each library involves different PCR reactions

So PCR duplicates can not occur between fragment of two different libraries.

But similar fragment could be found between libraries when the coverage is high.

solution_recal1

The vendors tend to inflate the values of the bases in the reads.

The recalibration tries to lower the scores of some biased motifs for some technologies.

solution_DOC1

Yes the mean coverage of the region is 36x:

```
summaryCoverageThreshold is a useful function to see if your coverage is uniform.
```

Another way is to compare the mean to the median:

```
If both are quite different that means something is wrong in your coverage.
```

note of mean vs median

solution_insert1

ERR seems to contain 2 types of libraries:

```
- PE fragments 195bp insert
- Mate pair fragments 2.3kb inserts
```

solution_alnMetrics1

Yes, 96% of the reads have been aligned Usually, we consider:

- A good alignment if $> 85\%$
- Reference assembly issues if [60-85]%
- Probably a mismatch between sample and ref if < 60 %

solution_snpeff1

We can see in the vcf that snpEff added a few sections

Mostly snpEff added predictions of the inpact of the variant based on known transcript positions (HIGH, MODERATE, LOW, MODIFIER, . . .)

solution_snpeff2

There is many tools you can use to explore vcf (vcftools, GEMIN . . .) but for such a simple task the easiest way is tu use the old good grep command:

```
grep "HIGH\|MODERATE"  pairedVariants/varscan.snpeff.vcf | grep SOMATIC | less
```

solution_igv1

some commands to find somatic variant in the vcf file

varscan

```
grep SOMATIC pairedVariants/varscan.vcf
```

MuTecT

```
grep -v REJECT pairedVariants/mutect.vcf | grep -v "^#"
```

Strelka

```
grep -v "^#" pairedVariants/strelka.vcf
```

Then look at some of these position in IGV