BCGES short courses, session 5: Picard tools, CNV analysis, BEDtools

Vincent Plagnol

Contents

1 A few more advanced ideas to consider (40 minutes)

1.1 Linux piping to limit input/output usage (10 minutes)

The lines of code below cannot be run as such but are just meant to give an example of the sort of things that multiple lines of code put together can do. In this example the only thing written to the disk is the sorted BAM file. No other intermediate file is written to disk.

```
novoalign -c 11 -o SAM -F STDFQ -f fasta1.fq fasta2.fq  -d $reference |
   samtools view - -u -S -b |
   novosort - -t /scratch0/ -c 1 -m 3G -i -o output_sorted.bam
```

Exercise: Understand the structure of the code and what the options used in the code above mean (at least for samtools which is freely available).

1.2 CRAM format as an alternative to BAM (10 minutes)

Recently (and after we finished preparing the software for this course), the `samtools` developers released version 1.0 of the software. A key addition is the handling of the CRAM format, an optimized format that compresses BAM further. Given the very large size of exome/genome datasets, CRAM has the potential to make a real difference. The version 1.0 of `samtools` has been installed in the bin subdirectory of session 5, and hopefully runs without any issue on your current machine. The main page for `samtools` has moved to a different location, and the associated documentation is here. I strongly recommend reading this very good blog post on the CRAM format that will tell you more about why this is useful.

Exercise: Take an example BAM file provided as part of this practical and convert it to CRAM. Do you see a reduction of the file size?

1.3 Tabix indexing of position sorted files (20 minutes)

The indexing technology used for the BAM format relies on the fact that the rows of a BAM file are indexed by chromosome and position. The same can be said of a VCF file for example. And as a consequence the same sort of tools can be used to index these files. A first place to look is the `tabix` man page, which highlights some of the ways `tabix` can be used.

Here is a basic example on a VCF file that you created in an earlier session.

```
bgzip  -c ../session4_multi_sample_calling_annotation/results/GATK_msamplecalling_HC.vcf \
    > results/GATK_msamplecalling_HC.vcf.gz
tabix -p vcf -s 1 -b 2 -e 2 results/GATK_msamplecalling_HC.vcf.gz

ls -ltrh results/GATK_msamplecalling_HC.vcf.gz*

tabix results/GATK_msamplecalling_HC.vcf.gz 21:43866166-43866166

## -rw-r--r--  1 vplagnol  staff   182B  2 Sep 02:14 results/GATK_msamplecalling_HC.vcf.gz.tbi
## -rw-r--r--  1 vplagnol  staff    14K  2 Sep 02:14 results/GATK_msamplecalling_HC.vcf.gz
## 21 43866166 . G C 567.21 . AC=7;AF=0.700;AN=10;BaseQRankSum=0.731;ClippingRankSum=0.731;DP=18;FS=3.358;
GT:AD:DP:GQ:PL ./.:0,0:0 ./.:0,0:0 1/1:0,2:2:6:90,6,0 ./.:0,0:0 1/1:0,4:4:12:165,12,0 1/1:0,4:4:12:180,12,
./.:0,0:0 ./.:0,0:0 ./.:0,0:0 0/0:3,0:3:0:0,0,90 0/1:1,4:5:97:165,0,97
```

Exercise: `tabix` may need to tweak some options to work with different files formats. The `R` code below creates a large comma separated file, with chromosome and position (two chromosomes, both 1 Mb long). Can you index the file and use `tabix` to retrieve the rows overlapping positions 1:100000-120000.

```
nrows <- 10000
chromosomes <- c(rep("chr1", times = 50000), rep("chr2", 50000))
start <- c(sort(sample(x = 10^6, size = 50000, replace = FALSE)), sort(sample(x = 10^6,
    size = 50000, replace = FALSE)))
end <- c(sort(sample(x = 10^6, size = 50000, replace = FALSE)), sort(sample(x = 10^6,
    size = 50000, replace = FALSE))) + sample(x = 1000, size = 1e+05, replace = TRUE)
random.number <- runif(10^5)
my.data <- data.frame(start = start, end = end, chromosome = chromosomes, very.important.number = random.n
```

```
write.table(x = my.data, file = "results/large_data_file.tab", quote = FALSE,
    col.names = TRUE, row.names = FALSE, sep = "\t")
```

Exercise: Another useful feature of `tabix` is that it also works on the web. Let us illustrate this with a simple example. Find the location of the latest 1KG VCF data, and retrieve all the variants located around the *BRCA1* gene.

2 Some examples PICARD tools (50 minutes)

2.1 Marking duplicates

Here is an example below of a typical PICARD call to mark duplicates. Make sure that you can run some version of this script, this is a very standard process to go through.

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
java -Xmx4g -jar /usr/local/gatk/picard.jar MarkDuplicates  ASSUME_SORTED=true \
      REMOVE_DUPLICATES=FALSE \
      INPUT=$BAM \
      OUTPUT=results/HG00130.mapped.bam METRICS_FILE=results/HG00130.metrics.out

## [Wed Sep 02 02:14:24 BST 2015] picard.sam.markduplicates.MarkDuplicates INPUT=[../data/BAM_files/HG0013
## [Wed Sep 02 02:14:24 BST 2015] Executing as vplagnol@Vincents-MacBook-Pro.local on Mac OS X 10.10.5 x86
## INFO 2015-09-02 02:14:25 MarkDuplicates Start of doWork freeMemory: 127018136; totalMemory: 128974848;
## INFO 2015-09-02 02:14:25 MarkDuplicates Reading input file and constructing read end information.
## INFO 2015-09-02 02:14:25 MarkDuplicates Will retain up to 14684096 data points before spilling to disk.
## WARNING 2015-09-02 02:14:25 AbstractOpticalDuplicateFinderCommandLineProgram Default READ_NAME_REGEX '[
## INFO 2015-09-02 02:14:25 MarkDuplicates Read 2699 records. 4 pairs never matched.
## INFO 2015-09-02 02:14:25 MarkDuplicates After buildSortedReadEndLists freeMemory: 67935816; totalMemory
## INFO 2015-09-02 02:14:25 MarkDuplicates Will retain up to 119308288 duplicate indices before spilling t
## INFO 2015-09-02 02:14:26 MarkDuplicates Traversing read pair information and detecting duplicates.
## INFO 2015-09-02 02:14:26 MarkDuplicates Traversing fragment information and detecting duplicates.
## INFO 2015-09-02 02:14:26 MarkDuplicates Sorting list of duplicate records.
## INFO 2015-09-02 02:14:26 MarkDuplicates After generateDuplicateIndexes freeMemory: 186065768; totalMemo
## INFO 2015-09-02 02:14:26 MarkDuplicates Marking 156 records as duplicates.
## INFO 2015-09-02 02:14:26 MarkDuplicates Found 0 optical duplicate clusters.
## INFO 2015-09-02 02:14:26 MarkDuplicates Before output close freeMemory: 1139829040; totalMemory: 114294
## INFO 2015-09-02 02:14:26 MarkDuplicates After output close freeMemory: 1139894440; totalMemory: 1142947
## [Wed Sep 02 02:14:26 BST 2015] picard.sam.markduplicates.MarkDuplicates done. Elapsed time: 0.03 minute
## Runtime.totalMemory()=1142947840
```

Exercise: Can you find where, in the headers of the resulting BAM file, how information is recorded about this duplicate marking step?

2.2 Collect summary statistics

And here is some example code to compute summary statistics on a BAM file:

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
java -Xmx4g -jar /usr/local/gatk/picard.jar CollectAlignmentSummaryMetrics \
    INPUT=$BAM \
    OUTPUT=results/insert_size.txt

## [Wed Sep 02 02:14:27 BST 2015] picard.analysis.CollectAlignmentSummaryMetrics INPUT=../data/BAM_files/H
## [Wed Sep 02 02:14:27 BST 2015] Executing as vplagnol@Vincents-MacBook-Pro.local on Mac OS X 10.10.5 x86
## [Wed Sep 02 02:14:27 BST 2015] picard.analysis.CollectAlignmentSummaryMetrics done. Elapsed time: 0.00
## Runtime.totalMemory()=128974848
```

Make sure that you can run that code, and look at the output files. Check that you understand the meaning of the summary statistics.

2.3 Convert a BAM file to fastq

Now something quite a bit more challenging: try the following script to convert a BAM file to fastq: Here is a first attempt

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
java -Xmx4g -jar /usr/local/gatk/picard.jar SamToFastq \
    INPUT=$BAM \
    FASTQ=results/read1.fq SECOND_END_FASTQ=results/read2.fq
```

```
## [Wed Sep 02 02:14:27 BST 2015] picard.sam.SamToFastq INPUT=../data/BAM_files/HG00130.mapped.ILLUMINA.bw
## [Wed Sep 02 02:14:27 BST 2015] Executing as vplagnol@Vincents-MacBook-Pro.local on Mac OS X 10.10.5 x86
## [Wed Sep 02 02:14:27 BST 2015] picard.sam.SamToFastq done. Elapsed time: 0.01 minutes.
## Runtime.totalMemory()=128974848
## To get help, see http://broadinstitute.github.io/picard/index.html#GettingHelp
## Exception in thread "main" htsjdk.samtools.SAMFormatException: SAM validation error: ERROR: Found 4 unp
##  at htsjdk.samtools.SAMUtils.processValidationError(SAMUtils.java:451)
##  at picard.sam.SamToFastq.doWork(SamToFastq.java:201)
##  at picard.cmdline.CommandLineProgram.instanceMain(CommandLineProgram.java:206)
##  at picard.cmdline.PicardCommandLine.instanceMain(PicardCommandLine.java:95)
##  at picard.cmdline.PicardCommandLine.main(PicardCommandLine.java:105)
```

The issue is that some reads do not have a mate, and that creates issues with the FASTQ files. We need a fix to deal with that error message. We can try the following, which should remove non-mapped reads:

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
samtools view -f 0x0001 -f 0x0002 -b -o results/only_paired.bam $BAM

java -Xmx4g -jar /usr/local/gatk/picard.jar SamToFastq INPUT=results/only_paired.bam \
           FASTQ=results/read1.fq SECOND_END_FASTQ=results/read2.fq

## [Wed Sep 02 02:14:28 BST 2015] picard.sam.SamToFastq INPUT=results/only_paired.bam FASTQ=results/read1.
## [Wed Sep 02 02:14:28 BST 2015] Executing as vplagnol@Vincents-MacBook-Pro.local on Mac OS X 10.10.5 x86
## [Wed Sep 02 02:14:28 BST 2015] picard.sam.SamToFastq done. Elapsed time: 0.00 minutes.
## Runtime.totalMemory()=128974848
## To get help, see http://broadinstitute.github.io/picard/index.html#GettingHelp
## Exception in thread "main" htsjdk.samtools.SAMFormatException: SAM validation error: ERROR: Found 2 unp
##  at htsjdk.samtools.SAMUtils.processValidationError(SAMUtils.java:451)
##  at picard.sam.SamToFastq.doWork(SamToFastq.java:201)
##  at picard.cmdline.CommandLineProgram.instanceMain(CommandLineProgram.java:206)
##  at picard.cmdline.PicardCommandLine.instanceMain(PicardCommandLine.java:95)
##  at picard.cmdline.PicardCommandLine.main(PicardCommandLine.java:105)
```

But it fails again. With only 2 unpaired mates this time, so things are getting better. We can find out what these are:

```
samtools view results/only_paired.bam | awk '{print $1}' | sort | uniq -c | sort -r | tail -5

##    2 SRR707198.10122
##    2 SRR707198.10048168
##    2 SRR707198.10047979
##    1 SRR707198.24062323
##    1 SRR707198.20187117
```

Now let us remove these problematic reads, but we have to do it manually. I could not find a better way to do this than what is shown below:

```
samtools view -h results/only_paired.bam |
       awk '{if (($1 != "SRR707198.24062323") && ($1 != "SRR707198.20187117")) print}' \
       > results/paired_fixed.sam

samtools view -b -S -o results/paired_fixed.bam results/paired_fixed.sam

## [samopen] SAM header is present: 86 sequences.
```

The second line above takes a SAM file input (option -S) and returns a BAM file (option -b). The resulting file can, at last, be converted into a FASTQ format:

```
java -Xmx4g -jar /usr/local/gatk/picard.jar SamToFastq INPUT=results/paired_fixed.bam \
                       FASTQ=results/read1.fq SECOND_END_FASTQ=results/read2.fq
```

```
## [Wed Sep 02 02:14:29 BST 2015] picard.sam.SamToFastq INPUT=results/paired_fixed.bam FASTQ=results/read1
## [Wed Sep 02 02:14:29 BST 2015] Executing as vplagnol@Vincents-MacBook-Pro.local on Mac OS X 10.10.5 x86
## [Wed Sep 02 02:14:29 BST 2015] picard.sam.SamToFastq done. Elapsed time: 0.00 minutes.
## Runtime.totalMemory()=128974848
```

Exercise: Can you find a better way to convert a BAM to fastq? I could not but I have not looked extensively.

3 BEDtools (30 minutes)

BEDtools utilities are a swiss-army knife of tools for a wide-range of genomics analysis tasks. Individually, none of these tools looks impressive, but together with `samtools` they can be used to achieve impressive tasks. I could simply point you to the excellent documentation of the software (and you certainly should have a look) but I will also highlight a few useful ideas.

3.1 Computing depth of coverage for targeted DNA sequencing experiments

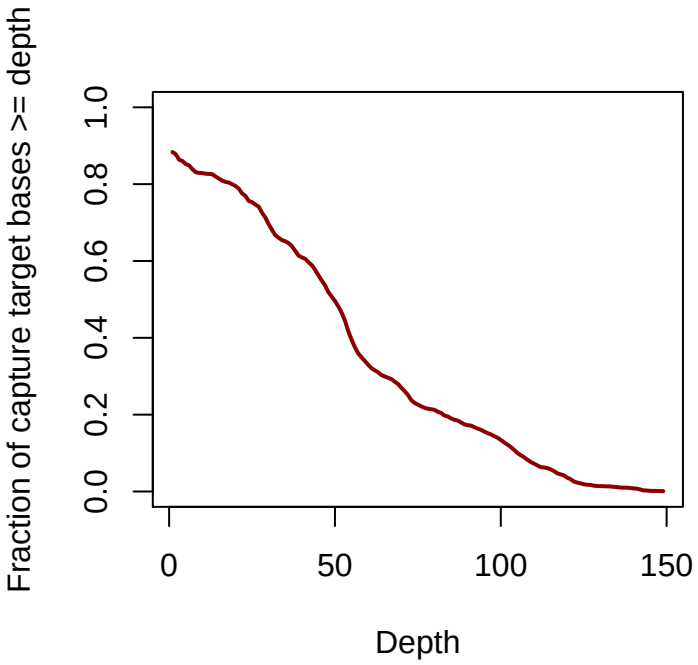
Some fancy bits of code are proposed in that blog post. We will simply follow these ideas to illustrate what `BEDtools` can do. The first step is to actually obtain a BED file that describes where the exons of a gene of interest are located. I generated this using this very good post, and if there is time I suggest that you look into this. My modified scripts (which worked in this case) are located in `tables/join.sh`.
Optional exercise: Reproduce the steps suggested in the Biostars answer. Scripts will need to be updated slightly depending on the exact computer that you use.

This is however not the main topic of the day and feel free to simply use the outcome BED file `tables/UBASH3A.bed`.

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
bedtools2.24 coverage -hist -b $BAM -a tables/UBASH3A.bed | grep ^all > results/HG00130_coverage.txt
head results/HG00130_coverage.txt

## all 0 284 2442 0.1162981
## all 1 14 2442 0.0057330
## all 2 35 2442 0.0143325
## all 3 8 2442 0.0032760
## all 4 20 2442 0.0081900
## all 5 8 2442 0.0032760
## all 6 24 2442 0.0098280
## all 7 19 2442 0.0077805
## all 8 5 2442 0.0020475
## all 9 1 2442 0.0004095
```

What you get from this is the data underlying an histogram. The fourth column tells you that you are looking at a total of 2,442 bp. For the first row, the third column tells you 284 of these positions are covered by 0 read, 14 are covered with 1 read, 35 positions with 2 reads... This is not very readable, so the output can for example be processed in `R` using the code below:

```
gcov = read.table("results/HG00130_coverage.txt");

# Create a cumulative distribution from the "raw" hist
# (truncate at depth >=1000)
gcov_cumul = 1 - cumsum(gcov[,5])

## Create a plot of the CDF
plot(x = gcov[2:143,2], y = gcov_cumul[1:142],
     col='darkred', type='l', lwd=2,
     xlab="Depth",
     ylab="Fraction of capture target bases >= depth",
     ylim=c(0,1.0))
```



Exercise: Make sure you can run and understand the `R` code, and what the different columns of the output mean. Can you rewrite the code above so that it loops over all the GBR BAM files? You will probably need the function `find` to do this. You can also try the fancier looking `parallel` implementation proposed by Stephen Turner's blog post but be careful, `parallel` is not systematically installed on all machines so thuis may not be possible.

3.2 Find regions that are not covered by a BAM file

This originated from the following question, asked on Twitter: "Given a.bam and b.regions.bed, how to get the parts of b.regions.bed that are not covered by a.bam?" See the example below:

```
BAM=../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam
regions=tables/UBASH3A.bed
bedtools2.24 genomecov -ibam $BAM -bga \
         | awk '$4==0' | bedtools2.24 intersect -a $regions -b - > results/not_covered.bed
```

Question: Understand what the three piping steps do in this example. Look at the output. How many bases are not covered? Is this consistent with the result file of the previous section?

4 Use `Rsamtools` to identify reads characteristic of a deletion (30 minutes)

Another strategy to detect CNVs consists of picking up reads that show an unusual pattern. For example, two pairs further apart than they normally should is potentially flagging a deletion. In this exercice I proposed to look at a well described 20 kb CNV located near the *IRGM* gene. We can start by looking at the location of the variant using the data from the Conrad et al paper. These are loaded into the ExomeDepth package, so this is one way to access them.

```
library(ExomeDepth)
data(Conrad.hg19)
IRGM <- Conrad.hg19.common.CNVs[ as.character(GenomicRanges::seqnames(Conrad.hg19.common.CNVs)) == "5" &
                                 GenomicRanges::start(Conrad.hg19.common.CNVs) > 150200000
                                 & GenomicRanges::end(Conrad.hg19.common.CNVs) < 150240000,]
write.table(x = as(IRGM, "data.frame"),
            row.names = FALSE, sep = '\t', quote = FALSE,
            file = 'IRGM_common.tab')
```

We now want to look for individuals that carry this variant. Low coverage whole genome data for the 1KG sample HG00123 will do the job.
Exercise: Identify the BAM file for mapped reads for HG00123 and download the slice chr5:150820438-150845438 (build hg38) using `samtools view`. How would you filter pairs of reads that span over the deletion in this sample?
Exercise: Use IGV to visualize the deletion and make sure you can clearly see these reads.
(Optional) Exercise: Can you write a R script (using `Rsamtools`) to identify these reads?

5 Read depth analysis to detect CNVs (30 minutes)

Another useful way to detect CNVs is read depth based analysis. This is a non straightforward analysis, and results can vary greatly between one dataset and the next. Hence, this part of the class is located at the end and can be skipped if there is not enough time. Nevertheless, it is a good idea to understand what this does and how to run an analysis like this one. To do so, look and run the example code provided in `exomeDepth_example.R`. Familiarise yourself with the code, modify it to run with the sample `UCLG_94_sorted_unique.bam`. You should be able to see a two exon heterozygous deletion located in the gene *GATA2*, which is causal in this individual. The code is listed below as well, but not executed on the fly as part of the document because it takes a bit long to do so. But there should be time to discuss what this does in class.

```
library(ExomeDepth)
```

```
ExomeCount <- read.table("../data/exon_read_count.tab.gz", header = TRUE)
my.test <- ExomeCount$UCLG.186_sorted_unique.bam
all.bams <- grep(pattern = 'bam$', names(ExomeCount), value = TRUE)
my.ref.samples <- subset( all.bams, all.bams != "UCLG.186_sorted_unique.bam")

my.reference.set <- as.matrix(ExomeCount[, my.ref.samples])

my.choice <- select.reference.set (test.counts = my.test,
                                   reference.counts = my.reference.set,
                                   bin.length = (ExomeCount$end - ExomeCount$start)/1000,
                                   n.bins.reduced = 10000)

my.matrix <- as.matrix( ExomeCount[, my.choice$reference.choice, drop = FALSE])
my.reference.selected <- apply(X = my.matrix,
                               MAR = 1,
                               FUN = sum)

######### create the ExomeDepth object
all.exons <- new("ExomeDepth",
                 test = my.test,
                 reference = my.reference.selected,
                 formula = 'cbind(test, reference) ~ 1')

all.exons <- CallCNVs(x = all.exons,
                      transition.probability = 10^-4,
                      chromosome = ExomeCount$chromosome,
                      start = ExomeCount$start,
                      end = ExomeCount$end,
                      name = ExomeCount$exons)


################## Now annotate the CNV calls
data(Conrad.hg19)
all.exons <- AnnotateExtra(x = all.exons,
                           reference.annotation = Conrad.hg19.common.CNVs,
                           min.overlap = 0.5,
                           column.name = "Conrad.hg19")

data(exons.hg19)

exons.hg19.GRanges <- GRanges(seqnames = exons.hg19$chromosome,
                              IRanges(start=exons.hg19$start,end=exons.hg19$end),
                              names = exons.hg19$name)

all.exons <- AnnotateExtra(x = all.exons,
                           reference.annotation = exons.hg19.GRanges,
                           min.overlap = 0.0001,
```

```
                            column.name = "exons.hg19")



####### Take the final table, sort by significance (Bayes factor)
my.final.table <- all.exons@CNV.calls
my.final.table <- my.final.table[ order(my.final.table$BF, decreasing = TRUE),]



print(table(is.na(my.final.table$Conrad.hg19)))

print(head(my.final.table))

pdf("figure/LRBA_example.pdf")
plot (all.exons,
      sequence = "4",
      xlim = c(151749334 - 100000, 151948710 + 100000),
      count.threshold = 20,
      main = "LRBA gene",
      with.gene = TRUE)

dev.off()
```