Handling sam and vcf data, quality control

We continue with the earlier analyses and get some new data:

```
cd ~/session_3

wget http://wasabiapp.org/vbox/data/session_4/file3.tgz
tar xzf file3.tgz

wget http://wasabiapp.org/vbox/data/session_4/file4.tgz
tar xzf file4.tgz
```

Note: In an earlier version of this document, the files above were called file1.tgz and file2.tgz. The names clashed with the names of the files from the previous session and "wget" renamed the new files as file1.tgz.1 and file2.tgz.1. You could then do:

```
mv file1.tgz.1 file1.tgz
tar xzf file1.tgz

mv file2.tgz.1 file2.tgz
tar xzf file2.tgz
```

Earlier we computed quality control statistics for fastq data:

```
# fastqc data/sample-1_1.fq.gz -o qc
# firefox qc/sample-1_1_fastqc.html
```

Now let's look at the quality of aligned bam data:

```
samtools stats data/sample-1_realn.bam  > qc/sample-1.bamstats

less qc/sample-1.bamstats

plot-bamstats -p qc/sample-1_bamstats qc/sample-1.bamstats
```

This gives an error. Fix your VM with command:

```
sudo apt-get update
sudo apt-get install gnuplot
```

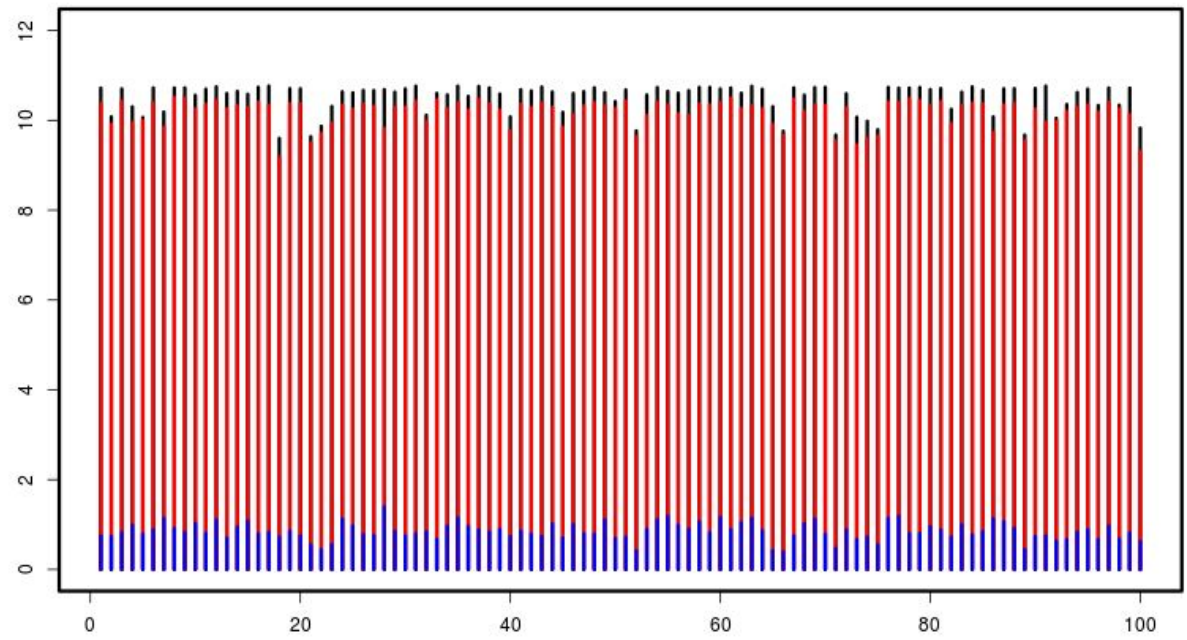
Password is "genomics". Try again.

```
firefox qc/sample-1_bamstats.html
```

The data that you have analysed are not "natural" as I have extracted only the reads that map to the small part of the genome that we are working with. Have a look on any file in

folder "bamstats_full". These are from real analysis and contain all data. Compare the files (your data set and one full data set) and see where the files differ.

Using the files in folder "bamstats_full" produce the following plot. Black, red and blue show the fields "total length", "bases mapped (cigar)" and "bases duplicated" from the bamstat files divided by the total length of the contigs. On y axis we have the 100 samples.



You can collect the numbers with a script like this:

```
cd ~/session_3/bamstats_full/

for i in `ls *stats`; do
   j=`basename $i .bam.stats`
   mapped=`awk '/bases mapped \(cigar\):/{print $5}' $i`
   duplicated=`awk '/bases duplicated:/{print $4}' $i`
   total=`awk '/total length:/{print $4}' $i`
   echo $j $total $mapped $duplicated
done > summary.txt
```

To get the length of the **full** reference genome, we need e.g. the fasta index file and a bit of awk:

```
wget http://wasabiapp.org/vbox/data/session_4/ninespine_full.fa.fai

awk '{t+=$2}END{print t}' bamstats_full/ninespine_full.fa.fai
```

Now plot the numbers as shown above. You can modify the script and add other statistics to collect and then plot.

Hint:

```
len=520934307
data = read.table("summary.txt")
colnames(data) = c("sample","tot","map","dup")
plot(1:100,data$tot/len,type="h",ylim=c(0,12))
lines(1:100,data$map/len,type="h",col="red")
lines(1:100,data$dup/len,type="h",col="blue")
```

We made the joint calling with our 20+1 samples. In reality, we have 100 samples, ten samples from ten different populations. In real analyses we would have called them all together.

Here we extract our target sample from the 20 other samples that were used for joint calling and then merge that with the 100 samples that I have previously analysed all together.

Extract one sample (see options "-s" and "-S" for `bcftools view`):

```
cd ~/session_3

bcftools view all.calls.vcf.gz -s sample-1 -Oz -o sample-1.vcf.gz

bcftools query -l all.calls.vcf.gz

bcftools query -l sample-1.vcf.gz
```

Compare the newly created **vcf** file to the **gvcf** file that we created last time:

```
bcftools view -H data/sample-1.calls.gvcf.gz | head

bcftools view -H sample-1.vcf.gz | head
```

Next, try the following command. It probably fails and you need to do something first. Repeat it then.

```
bcftools isec data/data100.vcf.gz sample-1.vcf.gz -p overlap
```

Read the file "overlap/README.txt". Count the number of variants in each category. (Remember to exclude the header!) Are you happy with the result?

Look at the command "bcftools merge".

```
bcftools merge
```

Merge the files "data100.vcf.gz" and "sample-1.vcf.gz" to a file called "data101.vcf.gz". Check the number of samples in the new file.

```
bcftools merge data/data100.vcf.gz sample-1.vcf.gz -Oz -o data101.vcf.gz

bcftools query -l data101.vcf.gz
```

Repeat masking

One central task for later analyses is to identify (and then remove) repeat regions. The most used for this is RepeatMasker (http://www.repeatmasker.org/). The software itself is free but it uses proprietary repeat libraries and distribution of those is forbidden. You can request them from the provider and typically get them free of charge for non-commercial usage. You should look at the RepeatMasker web-site for instructions for the installation of the software and the repeat libraries.

This is the command I used:

```
# RepeatMasker -xsmall -gff -dir ./data/rm -pa 10 -species "Gasterosteus aculeatus"
reference/ninespine.fa
```

Read the file "data/rm/RM_options.txt" to understand the command. Look then inside the other files in "data/rm". For us, the most important is the gff file. Google "gff ucsc" for more information.

---

bcftools doesn't support gff but it's easy to pick up the correct fields from a gff file and write it in a supported format:

```
awk '!/#/{OFS="\t";print $1,$4,$5}' data/rm/ninespine.fa.out.gff \
> data/rm/repeats.tab
```

With that, we can build up a command that does different kinds of filtering. Look at the bcftools options to understand what happens ("bcftools filter", "bcftools view")

```
bcftools filter -g 20 data101.vcf.gz | bcftools view -m2 -M2 -V mnps,indels \
-T ^data/rm/repeats.tab -Oz -o data101_clean.vcf.gz
```

Try the following the command. Fix the issue and do it again:

```
bcftools index -n data101_clean.vcf.gz

bcftools index -s data101_clean.vcf.gz | awk '{l+=$2;v+=$3}END{print l,v}'
```

What did you do? Do the same for "data101.vcf.gz". What do you learn?

About indexing commands

"samtools faidx" indexes fasta files and "bcftools index" vcf files. Both commands can also be used to get information from indexed data. "bcftools index -n" and "bcftools index -s" give different kinds of statistics. "samtools faidx <file> <region>" gives a sequence region. Compare the two commands:

```
samtools faidx reference/ninespine.fa ctg7180000010564:10-50

grep -A1 ctg7180000010564 reference/ninespine.fa

samtools faidx reference/ninespine.fa ctg7180000010564 | less
```

"samtools faidx <file>" and "samtools faidx <file> <region>" are useful commands for accessing specific regions in any large fasta file.

IGV

Integrative Genomics Viewer (IGV) is a popular viewer for genomic data.

1. Start it with command "igv".
2. Import our reference sequence "ninespine.fa" with "Genomes" -> "Load genome from file". Note that the sequence has to be indexed.
3. Import other data with "File" -> "Load from file"
    a. import variation data "data101_clean.vcf.gz"
    b. import alignment file "sample-1_realn.bam"
    c. import repeat annotation "ninespine.fa.out.gff"

You can resize the panels by dragging the edges. You can also compact the contents vertically by right-clicking the panel in the left and selecting "squished".

Finding and removing high-coverage regions/SNPs. Selecting SNPs based on MAF or missing data.

```
bcftools query -f '%CHROM,%POS,%DP\n' data101_clean.vcf.gz > dp_data101_clean.csv
```

Run in R. What is this?

```
data=read.csv("dp_data101_clean.csv",header=F)

hist(data$V3,breaks=1000)
hist(data$V3,breaks=1000,xlim=c(0,2000))

dim(data)
len=dim(data)[1]

res=aggregate(cbind(data$V3,rep(1,len)),by=list(data$V1),"sum")
good.ctg = res$Group.1[(res$V1/res$V2>400 & res$V1/res$V2<1800)]
```

```
length(good.ctg)

good.snps=data[data$V1 %in% good.ctg & data$V3>400 & data$V3<1800,c(1,2,2)]
write.table(good.snps,"good_snps.txt",sep="\t",quote=F,row.names=F,col.names=F)
```

Back to console:

```
bcftools view -T good_snps.txt data101_clean.vcf.gz -Oz -o data101_good.vcf.gz

bcftools index data101_good.vcf.gz
bcftools index -n data101_good.vcf.gz

bcftools index -n data101_clean.vcf.gz
```

What did you do? Bring the new variant file to IGV and compare differences. How can you easily find differences between two vcf files?

We start using a new program called "vcftools". It is versatile but somewhat slow has documentation on-line only. Vcftools can read .vcf.gz files but its decompression is very slow. We therefore use "bcftools" to decompress vcf.gz to vcf, pipe that to "vcftools" and then compress the output vcf back to vcf.gz using "bcftools". These three steps are separated by pipes | and "vcftools" is used with options "`--vcf -`" (read from stdin) and "`--stdout`" (write to stdout).

We first extract sites that have data for at least 50% of samples (`--max-missing 0.5`) and that have minor allele frequency of at least 5% (`--maf 0.05`):

```
bcftools view data101_good.vcf.gz | vcftools --vcf - --max-missing 0.5 --maf 0.05
--recode --recode-INFO-all --stdout | bcftools view -Oz -o data101_select1.vcf.gz
```

See what was the effect of this:

```
bcftools index data101_select1.vcf.gz
bcftools index -n data101_select1.vcf.gz
```

We then remove sites that are not in Hardy-Weinberg equilibrium:

```
bcftools view data101_select1.vcf.gz | vcftools --vcf - --hwe 0.000001 --recode
--recode-INFO-all --stdout | bcftools view -Oz -o data101_select2.vcf.gz

bcftools index data101_select2.vcf.gz
bcftools index -n data101_select2.vcf.gz
```

We can compute statistics how all this filtering has affected the set of data:

```
mkdir stats
bcftools stats data101.vcf.gz > stats/data101.stats
bcftools stats data101_select2.vcf.gz > stats/data101_select2.stats
```

```
less -S stats/data101.stats

grep ^TSTV stats/data101.stats
grep ^TSTV stats/data101_select2.stats
```

On human data, ts/tv is considered a good indicator if data are "good". Genome-wide ratio is around 2.0-2.1 but it can be closer to 3 in exome data. If transitions and transversions would be equally likely, the ratio would be 0.5. (Why?) For our data, the ratio is lower than for humans but our cleaning has changed the ratio to the expected direction.

One final thing we can do is to remove SNPs that are in tight linkage. As the data are not phased, we can't compute true linkage but we can use r^2 as proxy:

```
mkdir filter
bcftools view data101_select2.vcf.gz | vcftools --vcf - --geno-r2 --ld-window 10
--out filter/data101_select2
```

See the distribution in R:

```
d = read.table("filter/data101_select2.geno.ld",header=T)
hist(unlist(d$R.2))
```

Let's decide to use 0.2 as threshold and remove anything above that. In console:

```
bcftools view data101_select2.vcf.gz | vcftools --vcf - --geno-r2 --ld-window 10
--min-r2 0.2 --out filter/data101_select2

awk '!/CHR/{OFS="\t";print $1,$2,$2}' filter/data101_select2.geno.ld | uniq >
filter/data101_select2_r2.tab

bcftools view -T ^filter/data101_select2_r2.tab -Oz -o data101_select3.vcf.gz
data101_select2.vcf.gz

bcftools index data101_select3.vcf.gz
bcftools index -n data101_select3.vcf.gz
```

Bring the last variant file to IGV and compare differences to the earlier one.

A recap of things learned this far

From fastq to vcf

We have analysed one sample from raw sequencing data (fastq or .fq.gz) all the way to filtered variant calls. The steps taken were:

1. mapping of fastq data against a reference genome (in our case, 200 contigs): the output is a sam-formatted alignment file that we convert to binary bam-format.

2. sorting of bam alignment: in the original bam file, the reads are in the input order (as in the fastq files) and they have to be sorted according to their mapped coordinates in the reference genome.
3. removal of duplicates: these are from PCR amplification and not truly representing the variation in the data.
4. realignment of reads around indels to correct for the inconsistencies in the mapped data.
5. calling of variants in genomic vcf (gvcf) format: in contrast to a regular vcf, gvcf also contains information about the non-variant sites and allows to distinguish (a) sites that are identical with the reference from (b) sites for which we have no information. (in IGV, these are indicated by white and gray background color.)
6. joint-calling of multiple gvcf files to generate a vcf file that contains all the samples as one big table.
7. filtering of variant calls.

We focus on binary SNP variants. The steps we did to filter the data were:

1. extraction of our sample from the original 20+1 vcf file, producing a vcf file containing only that sample.
2. merge of that one-sample vcf file with a vcf file containing 100 samples called independently.
3. removal of variants (a) closer than 20 bases from the closest indel, (b) having fewer than two or more than two alleles, (c) being not of SNP-type, and (d) overlapping inferred repeat elements.
4. removal of (a) contigs and (b) variants that have lower than expected or higher than expected sequencing coverage.
5. removal of variants that have more than 50% missing data or minor allele frequency under 5%.
6. removal of variants that are not in Hardy-Weinberg equilibrium.
7. removal of variant that are in tight linkage with neighbouring variants.

We will next study the effects of this filtering for downstream analyses.

Vcf format and bcftools

A vcf file is a big table where the variant positions make the rows and the samples make the columns. The table starts with metainformation and a header defining the columns.

A central tool for handling the vcf files is "bcftools" and its many subcommands. Some functions we have used:

- view the data: -H prevents printing the metainformation and the header

```
bcftools view -H data101_select1.vcf.gz | less -S
```

- view the data of one sample only: with -s a list of samples, with -S a file with sample names

```
bcftools view -H -s sample-1 data101_select1.vcf.gz | less -S
```

- view the data in a nicer format using "bcftools query": with -f we define the format, see http://www.htslib.org/doc/bcftools.html for details

```
bcftools query -s sample-1 -f '%CHROM %POS %REF %ALT  [SAMPLE=%GT]\n' \
data101_select1.vcf.gz | less -S
```

We can use "bcftools query" and "samtools tview" to see the connection between the variant calls and the alignment data:

```
bcftools query -s sample-1 -f '%CHROM %POS %REF %ALT  [%SAMPLE=%GT]\n' \
data101_select1.vcf.gz | grep "0/0" | head -3

ctg7180000005484 19163 A G  sample-1=0/0
ctg7180000005484 19194 G A  sample-1=0/0
ctg7180000005484 19578 G C  sample-1=0/0
```

Here, we have three variant positions where our sample is homozygote (0/0) for the first allele (0). The first two columns tell the contig (or chromosome) name and position; the third and fourth column list the reference and variant allele; and the fifth column tells the genotype for our sample. We can use "samtools tview" to show the locus in the alignment data:

```
samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:19163
```

Note that with option -p we can give the position to jump to and do not need to scroll so much within the tview data browser.

We can do the same for heterozygote positions (0/1):

```
bcftools query -s sample-1 -f '%CHROM %POS %REF %ALT  [%SAMPLE=%GT]\n' \
data101_select1.vcf.gz | grep "0/1" | head -3

ctg7180000005484 19022 G T  sample-1=0/1
ctg7180000005484 19030 T A  sample-1=0/1
ctg7180000005484 19172 C G  sample-1=0/1

samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:19022
```

and homozygote variant positions (1/1):

```
bcftools query -s sample-1 -f '%CHROM %POS %REF %ALT  [%SAMPLE=%GT]\n' \
data101_select1.vcf.gz | grep "1/1" | head -3

ctg7180000005484 35194 A C  sample-1=1/1
```

```
ctg7180000005484 35572 G A  sample-1=1/1
ctg7180000005484 38228 T C  sample-1=1/1

samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:35194
```

Note that the coordinates of variant positions may be different in your sample and you may need to edit the position ("-p contig:position") in the "samtools tview" commands. For my sample, the bam data look roughly like this:

homozygote reference: position 19163

```
samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:19153 -d T | cut -c -30

       19161     19171     19
ATGTTGTGCCAGAAACAATCATTTTGTCTG
...................S..........
.................  ,,,,,,,,,,
...................G      ,,,,
........................
...................G..........
..............................
.........T....C....G..........
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
...................G..........
...................G..........
...................G..........
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
,,,,,,,,,,,,,,,,,,g,,,,,,,,,
```

Note that the reference allele is at position **19163**; position 19172 contains another variant for which our sample is heterozygote:

```
ctg7180000005484 19172 C G  SAMPLE=0/1
```

heterozygote: positions 19022 and 19030

```
samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:19012 -d T | cut -c -30

         19021     19031     1
ATGTATCGCAGGAAAAAATGAGTGACCACA
..........K.......W...........
..........T.......A...........
..........T.......A...........
```

```
..........T.......A..........
,,,,,,,,,,t,,,,,,,a,,,,,,,,,,
..........T.......A..........
............................
............................
         ..T.......A..........
            .......A..........
                  ............
```

homozygote variant: position 35194

```
samtools tview data/sample-1_realn.bam reference/ninespine.fa \
-p ctg7180000005484:35184 -d T | cut -c -30
```

```
       35191     35201     352
AAACACAGGAAGAATACTGACAGGCCAAAG
..........C...................
..       .C...................
,,,,,,,,,,c,,,,,,,,,,,,  ....
,,,,,,,,,,c,,,,,,,,,,,,,,,,,,
..........C...................
..........C...................
..........C...................
..........C...................
..........C...................
..........C...................
..........C...................
..........C...................
,,,,,,,,,,c,,,,,,,,,,,,,,,,,,
..........C...................
..........C...................
,,,,,,,,,,c,,,,,,,,,,,,,,,,,,
,,,,,,,,,,c,,,,,,,,,,,,,,,,,,
..........C...................
```