Genomics

Freie Universität Berlin, Institut für Informatik
Knut Reinert, Peter Robinson, Sebastian Bauer
Wintersemester 2012/2013

1. Übungsblatt vom 18. Oktober 2012
Diskussion am 25. Oktober 2012

Workflow

For this exercise, you will use a number of tools to perform pileup analysis. One practical difficulty of such analyses lies in the very size of the files typically encountered in NGS work. In order to keep things reasonably easy, we will work with a reduced exome file. In order for you to get some practice in the use of these tools, we supply the detailed workflow in the following.

File download

We will use the BAM file from the following article: Glusman G et al. (2012) Low budget analysis of Direct-To-Consumer genomic testing familial data. *F1000Research*, **1**:3[1] Search for Son's Exome Files in this document and download the BAM file (ca. 4 Gb).

The file is called `Son_s_Aligned_Bam_File.bam`.

Software download

I downloaded the most recent versions of samtools

- samtools (`samtools-0.1.19.tar.bz2`) from http://sourceforge.net/projects/samtools/, *install as tar xvfj samtools-0.1.19.tar.bz2; cd samtools-0.1.19; make*. The following assumes that the samtools executables are visible in your PATH.[2]

Extract a subset of the BAM file

We will extract the reads from the smallest chromosome (chr21) for all downstream analyses. To do so, we need to sort and index the BAM file:

```
$ samtools sort Son_s_Aligned_Bam_File.bam Son.sorted
$ samtools index Son.sorted.bam
```

The first command produces a sorted BAM file called `Son.sorted.bam`, and the second command produces an index called `Son.sorted.bam.bai`. On my desktop computer, the entire process took about ten minutes.

To extract a subset of this information, we need to do the following.

```
$  samtools view Son.sorted.bam  "21:31540484-34532683" > Son.sorted.chr21subset.sam
```

This will extract all reads that map to an area of about 3 million base pairs on chromosome 21 (chr21:31,540,484-34,532,683). To recreate a BAM file, we need to re-extract the header from the original BAM file, append it to the subsetted SAM file, and reconvert back to a BAM file:

```
$ samtools view -H Son.sorted.bam > header.txt
$ cat header.txt Son.sorted.chr21subset.sam > Son.chr21subsetWithHeader.sam
$ samtools view -Sb Son.chr21subsetWithHeader.sam > Son.chr21.bam
```

[1] (`http://f1000research.com/articles/1-3`).

[2] I cc'd into the directory with the BAM file and entered: $ export PATH=$PATH:/home/peter/bin/samtools-0.1.19/

In the final command, the ‘‘-S’’ indicates the input is in SAM format and the ‘‘b’’ indicates that you'd like BAM output. For those of you who may have difficulty performing the above analysis, I have put the file `Son.chr21.bam` (4.5 Mb)into my Dropbox and will send you an invitation if you send me an email.

Exercise 1.

Starting with the "small" BAM file we created above, sort and index the file, convert it to a SAM file, and then use some Linux tools to explore the data in the file. Note that we will be using the GNU version of awk, which includes various functions for analysing bitfields. With kubuntu, this can be installed with the command `$ sudo apt-get install gawk`. The program gawk can then be executed via the commands `$ gawk$` or simply `$ awk$`.

```
$ samtools sort Son.chr21.bam Son.chr21.sorted
$ samtools index Son.chr21.sorted.bam
$ samtools view Son.chr21.sorted.bam > Son.chr21.sorted.sam
```

To do this exercise, we will use some shell tools to explore the sam file. Please consult the online description of the SAM format before doing this exercise. We will explain a more detailed example here to get you started.

Let's say we want to find out how many reads were perfect matches over all 90 bases. We consult the online documentation for SAM format and discover the the sixth field contains a CIGAR string (recall this from lecture 1). The CIGAR representation of 90 matches is simply 90M. To see the cigar field in the SAM file, we use the cut command:

```
$ cut -f 6 Son.chr21.sorted.sam
```

We now pipe the results of this command into awk, which is a useful tool for text processing and data extraction that was one of the inspirations for Larry Wall while he was creating perl.

```
$ cut -f 6 Son.chr21.sorted.sam | awk '/90M/ {C++} END { print C }'
```

This command first extracts the sixth field of the SAM file, pipes it into awk, which in turn executes the expression `C++` every time a line matches the regular expression `90M`. At the end of the file, it then prints the value of `C` (33043). To find out more about awk, Google, as well as the unix "man" command, are your best friends.

The second field of a SAM file represents a bitwise flag. I have copied the table from the SAM documentation in the following:

Bit	Description
0x1	template having multiple segments in sequencing
0x2	each segment properly aligned according to the aligner
0x4	segment unmapped
0x8	next segment in the template unmapped
0x10	SEQ being reverse complemented
0x20	SEQ of the next segment in the template being reversed
0x40	the first segment in the template
0x80	the last segment in the template
0x100	secondary alignment
0x200	not passing quality controls
0x400	PCR or optical duplicate
0x800	supplementary alignment

The following command will print the second field of each line:

```
$ awk '{print $2}'   Son.chr21.sorted.sam
```

Now we can ask questions such as how many reads were called as "PCR or optical duplicate"

```
 awk 'and($2,0x0400)'   Son.chr21.sorted.sam  | wc -l
```

Note that the "and" function performs a logical AND operation on the bit string represented by the integer in the FLAG field[3]

For the exercise, calculate how many reads are on the forward strand, and how many are on the reverse strand:

Exercise 2.

a) How many reads were mapped?

b) How many reads were mapped to the reverse strand?

c) Now calculate how many reads begin between position 31,000,000 and 32,500,000 on chromosome 21. If you have trouble with this consult Google about if-else syntax for awk.

Exercise 3.

Now we will generate a pileup file with samtools. We first need to download the reference file, which is the human chromosome 21 from build hg19. Download the corresponding file from the UCSC genome browser[4]. Note that this file uses "chr21" as the name of the chromosome, but the BAM file we are working with simply uses "21" (chr21.fa.gz). This means we need to change the name of the sequence in the FASTA file to make sure samtools will recognize that it is the same.

```
$ echo ">21" > chr21.fa
$ gunzip -c chr21.fa.gz | sed '1d' >> chr21.fa
```

The first command creates a file with the single line `>21`. The second line unzips the gzipped file we downloaded from UCSC and sends the output to STDOUT, which is then piped through a sed command that deletes the first line (i.e., `>chr21`) and appends the rest of the file to the chr21.fa file created by the first line. We now need to create an index file for this FASTA file.

```
$ samtools faidx chr21.fa
```

This creates an index file called `chr21.fa.fai` Let us now create the pileup file.

```
$ samtools mpileup -f chr21.fa Son.chr21.sorted.bam > Son.chr21.pileup
```

Use the tview program of samtools to compare the alignment of the pileup file with tview

```
$ samtools tview Son.chr21.sorted.bam chr21.fa
```

[3] See this website for an introduction: https://www.gnu.org/software/gawk/manual/html_node/Bitwise-Functions.html

[4] http://hgdownload.soe.ucsc.edu/goldenPath/hg19/chromosomes/

You will need to tell samtools what part of the alignment you would like to see. Since this is an exome file, most of the chromosome has little or no reads aligned to it. You can peak into the pileup file to find a place with enough reads to make the alignment interesting. Then, press the "g" key, and you can enter the position as (for instance) `21:31691775`. Look at individual columns and compare the alignment with the pileup and make sure you understand what the format means.

For the exercise, you are asked to now write a script or program in a language of your choice (e.g., C, Java, Perl, Python, FORTRAN, or LISP) to produce data regarding the coverage of the exome at all positions with at least one read. Your program or script should output the data in the following format

```
1 x
2 y
3 z
4 ...
```

That is, there were x positions with a coverage of 1 read, y positions with a coverage of 2 reads, and so on. For extra credit, use R to plot your results.

Exercise 4.

Write a script or program in a language of your choice (see above) to perform simple variant calling using the pileup file. Your program should implement the rules we discussed in the lecture

- Determine if both samples meet the minimum coverage requirement (by default, three reads with base quality ≥ 20)
- Determine a genotype for each sample individually based upon the read bases observed. By default, a variant allele must be supported by at least two independent reads and at least 8% of all reads.
- Variants are called homozygous if supported by 75% or more of all reads at a position; otherwise they are called heterozygous.