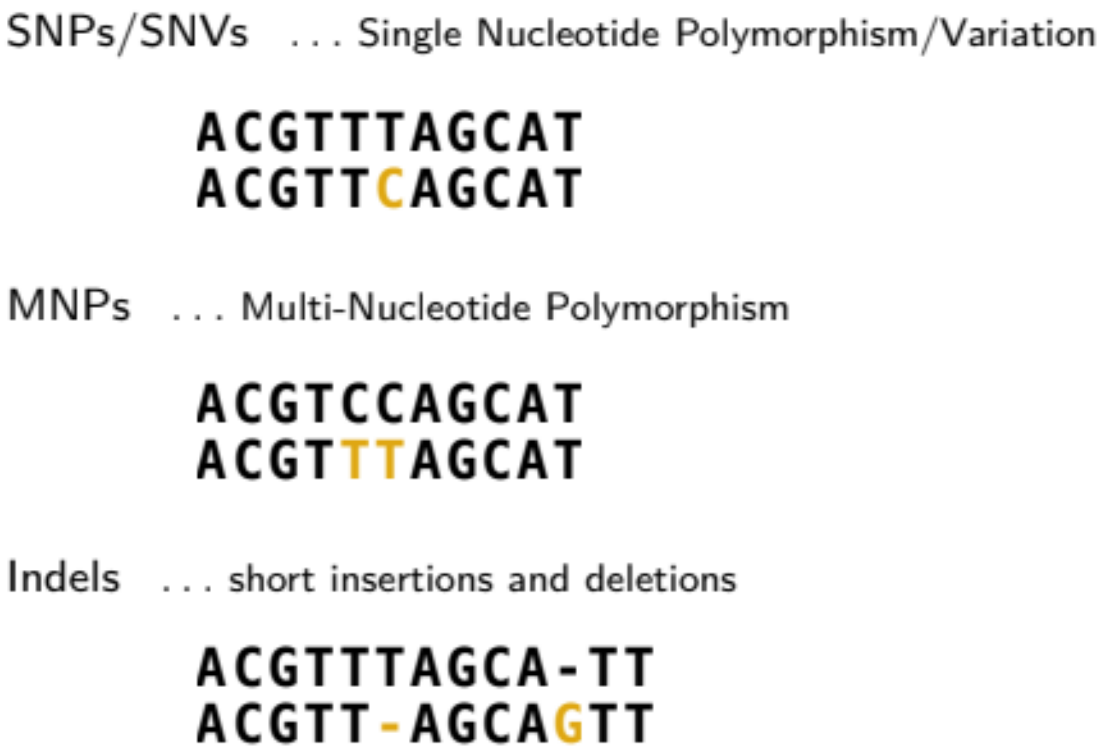
1 Variant Calling

1.1 Introduction

Variant calling is the process of identifying differences between the reference genome and the samples that have been sequenced. These differences can be single nucleotide polymorphisms (SNPs), multi-nucleotide polymorphisms (MNPs) or small insertions and deletions (indels) and examples of each of these are shown below.



1.2 Learning outcomes

On completion of the tutorial, you can expect to be able to:

- Perform variant calling (SNPs and indels) using standard tools
- Assess the quality/confidence of a variant call
- Filter variant calls to remove low quality/confidence calls
- Perform variant calling across multiple samples
- Visualise variants using standard tools
- Annotate variants with consequence calls

1.3 Tutorial sections

This tutorial comprises the following sections:
1. Performing variant calling 2. Filtering variants 3. Multi-sample variant calling 4. Visualising variants

There is also an additional (optional) section: 5. Variant annotation

1.4 Authors

This tutorial was written by Jacqui Keane based on material from Thomas Keane and Petr Danecek.

1.5 Running the commands from this tutorial

You can follow this tutorial by typing all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, open a new terminal on your computer and type the command below:

[ ]:
```
cd ~/course_data/variant_calling/data
```

Now you can follow the instructions in the tutorial from here.

1.6 Let's get started!

This tutorial assumes that you have samtools, bcftools and IGV installed on your computer. These are already installed on the VM you are using. To check that these are installed, you can run the following commands:

[ ]:
```
samtools --help
```

[ ]:
```
bcftools --help
```

This should return the help message for samtools and bcftools.

To get started with the tutorial, go to to the first section: Performing variant calling

2 Performing Variant Calling

When performing variant calling we need the aligned sequences in SAM, BAM or CRAM format and the reference genome that we want to call varants against.

First, check you are in the correct directory.

[ ]:
```
pwd
```

It should display something like:

`/home/manager/course_data/variant_calling/data`

2.1 Assessing the input data

To list the files in the current directory, type

[ ]:
```
ls -lh
```

The listing shows aligned data for two mouse strains A/J and NZO (A_J.bam and NZO.bam) and the chromosome 19 of the mouse reference genome (GRCm38_68.19.fa).

Before performing variant calling, it is important to check the quality of the data that you will be working with. We have already seen how to do this in the QC and Data Formats and Read Alignment sessions. The commands would look like:

```
samtools stats -r GRCm38_68.19.fa A_J.bam > A_J.stats

samtools stats -r GRCm38_68.19.fa NZO.bam > NZO.stats

plot-bamstats -r GRCm38_68.19.fa.gc -p A_J.graphs/ A_J.stats

plot-bamstats -r GRCm38_68.19.fa.gc -p NZO.graphs/ NZO.stats
```

You do not need to run these QC checks on this data and for this we will assume that QC has already been performed and the data is of good quality.

2.2 Generating pileup

The command `samtools mpileup` prints the read bases that align to each position in the reference genome. Type the command:

[ ]:
```
samtools mpileup -f GRCm38_68.19.fa A_J.bam | less -S
```

Each line corresponds to a position on the genome.

The columns are: chromosome, position, reference base, read depth, read bases (dot . and comma , indicate match on the forward and on the reverse strand; ACGTN and acgtn a mismatch on the forward and the reverse strand) and the final column is the base qualities encoded into characters. The caret symbol ^ marks the start of a read (followed by the mapping quality encoded as an ASCII character), the dollar sign $ the end of a read, deleted bases are represented by asterisk *.

This output can be used for a simple consensus calling. One rarely needs this type of output. Instead, for a more sophisticated variant calling method, see the next section.

2.2.1 Exercises

Look at the output from the `mpileup` command above and answer the following questions.

Q1: What is the read depth at position 10001994? (Rather than scrolling to the position, use the substring searching capabilities of less: press /, then type 10001994 followed by enter to find the position.)

Q2: What is the reference allele and the alternate allele at position 10001994?

Q3: How many reads call the reference allele at position 10001994 and how many reads call the alternate allele at position 10001994?

2.3 Generating genotype likelihoods and calling variants

The `bcftools mpileup` command can be used to generate genotype likelihoods. (Beware: the command mpileup is present in both samtools and bcftools, but in both they do different things. While `samtools mpileup` produces the text pileup output seen in the previous exercise, `bcftools mpileup` generates a VCF file with genotype likelihoods.)

Run the following command (when done, press q to quit the viewing mode):

[ ]:
```
bcftools mpileup -f GRCm38_68.19.fa A_J.bam | less -S
```

This generates an intermediate output which contains genotype likelihoods and other raw information necessary for variant calling. This output is usually streamed directly to the caller like this

```
bcftools mpileup -f GRCm38_68.19.fa A_J.bam | bcftools call -m | less -S
```

The output above contains both variant and non-variant positions. Check the input/output options section of the `bcftools call` usage page and see if there is an option to print out only variant sites. Then construct a command to print out variant sites only:

[ ]:
```

```

The INFO and FORMAT fields of each entry tells us something about the data at the position in the genome. It consists of a set of key-value pairs with the tags being explained in the header of the VCF file (see the ##INFO and ##FORMAT lines in the header).

We can tell mpileup to add additional ##INFO and ##FORMAT information to the output. For example, we can ask it to add the FORMAT/AD tag which informs about the number of high-quality reads that support alleles listed in REF and ALT columns. The list of all available tags can be printed with the command:

[ ]:
```
bcftools mpileup -a ?
```

Now let's run the variant calling again, this time adding the `-a AD` option. We will also add the `-Ou` option so that it streams a binary uncompressed BCF into call. This is to avoid the unnecessary CPU overhead of formatting the internal binary format to plain text VCF only to be immediately formatted back to the internal binary format again.

[ ]:
```
bcftools mpileup -a AD -f GRCm38_68.19.fa A_J.bam -Ou | bcftools call -mv -o␣
↪out.vcf
```

2.3.1 Exercises

Look at the content of the VCF file produced above and answers the questions that follow.

[ ]:
```
less -S out.vcf
```

Q1: What is the reference allele and the alternate allele at position 10001994?

Q2: What is the total raw read depth at position 10001994?

Note: This number may be different from the values we obtained earlier, because some low quality reads or bases might have been filtered previously.

Q3: What is the number of high-quality reads supporting the SNP call at position 10001994? How many reads support the reference allele and how many support the alternate allele?

Hint: Look up the AD tag in the FORMAT column: the first value gives the number of reads calling the reference allelle and the second gives the number of reads calling the alternate alleles.

Q4: What sort of event is happening at position 10003648?

Congratulations, you have sucessfully called variants from some NGS data. Now continue to the next section of the tutorial: filtering variants

3 Variant Filtering

In the next series of commands we will learn how to extract information from VCFs and how to filter the raw calls. We will use the bcftools commands again. Most of the commands accept the `-i, --include` and `-e, --exclude` options https://samtools.github.io/bcftools/bcftools.html#expressions which will be useful when filtering using fixed thresholds. We will estimate the quality of the callset by calculating the ratio of transitions and transversions https://en.wikipedia.org/wiki/Transversion.

When drafting commands, it is best to build them gradually. This prevents errors and allows you to verify that they work as expected. Let's start with printing a simple list of positions from the VCF using the bcftools query command https://samtools.github.io/bcftools/bcftools.html#query and pipe through the head command to limit the printed output to the first few lines:

```
[ ]: bcftools query --format 'POS=%POS\n' out.vcf | head
```

As you can see, the command expanded the formatting expression `POS=%POS\n` in the following way: for each VCF record the string POS= was copied verbatim, the string `%POS` was replaced by the VCF coordinate stored in the POS column, and then the newline character `\n` ended each line. (Without the newline character, positions from the entire VCF would be printed on a single line.)

Now add the reference and the alternate allele to the output. They are stored in the REF and ALT column in the VCF, and let's separate them by a comma:

```
[ ]: bcftools query -f'%POS %REF,%ALT\n' out.vcf | head
```

In the next step add the quality (%QUAL), genotype (%GT) and sequencing depth (%AD) to the output. Note that FORMAT tags must be enclosed within square brackets [...] to iterate over all samples in the VCF. (Check the Extracting per-sample tags section in the manual https://samtools.github.io/bcftools/howtos/query.html for a more detailed explanation why the square brackets are needed.)

```
[ ]: bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' out.vcf | head
```

Now we are able to quickly extract important information from the VCFs. Now let's filter rows with QUAL smaller than 30 by adding the filtering expression –exclude 'QUAL<30' or –include 'QUAL>=30' like this:

```
[ ]: bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' -i'QUAL>=30' out.vcf | head
```

Now compare the result with the output from the previous command, were the low-quality lines removed?

In the next step limit the output to SNPs and ignore indels by adding the type="snp" condition to the filtering expression. Because both conditions must be valid at the same time, we request the AND logic using the && operator:

```
[ ]: bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' -i'QUAL>=30 && type="snp"'␣
  ↪out.vcf | head
```

3.1 Exercises

Q1: Can you print SNPs with QUAL bigger than 30 and require at least 25 alternate reads in the AD tag?

Remember, the first value of the AD tag is the number of reference reads, the second is the number of alternate reads, therefore you will need to query the second value of the AD tag. The first value can be queried as AD[0] and the second as AD[1] (the allele indexes are zero-based). In case of FORMAT fields, also the queried sample must be selected as AD[sample:subfield] . Therefore add to the expression the condition AD[0:1] >= 25 to select the first (and in our case the only one) sample or AD[*:1] >= 25 to select any sample for which the condition is valid.

Now we can filter our callset. In order to evaluate the quality, we will use bcftools stats to calculate the ratio of transitions vs transversions. We start by checking what is the ts/tv of the raw unfiltered callset. The `stats` command produces a text output, we extract the field of interest as follows:

```
[ ]: bcftools stats out.vcf | less
     bcftools stats out.vcf | grep TSTV
     bcftools stats out.vcf | grep TSTV | cut -f5
```

Q2: Calculate ts/tv of the set filtered as above by adding -i 'QUAL>=30 && AD[*:1]>=25' to the bcftools stats command. (Here the asterisk followed by a colon tells the program to apply the filtering to all samples. At least one sample must pass in order for a site to pass.) After applying the filter, you should observe an increased ts/tv value.

Q3: Can you do the reverse and find out the ts/tv of the removed sites? Use the `-e` option instead of `-i`. The ts/tv of the removed low-quality sites should be lower.

Q4: The test data come from an inbred homozygous mouse, therefore any heterozygous genotypes are most likely mapping and alignment artefacts. Can you find out what is the ts/tv of the heterozyous SNPs? Do you expect higher or lower ts/tv? Use the filtering expression `-i 'GT="het"'` to select sites with heterozygous genotypes.

Another useful command is `bcftools filter` which allows to "soft filter" the VCF: instead of removing sites, it can annotate the FILTER column to indicate sites which fail. Apply the above filters ('QUAL>=30 && AD[*:1]>=25') to produce a final callset, adding also the `--SnpGap` and the `--IndelGap` option to filter variants in close proximity to indels:

```
[ ]: bcftools filter -s LowQual -i'QUAL>=30 && AD[*:1]>=25' -g8 -G10 out.vcf -o out.
  ↪flt.vcf
```

3.2 Variant normalization

The same indel variant can be represented in different ways. For example, consider the following 2bp deletion. Although the resulting sequence does not change, the deletion can be placed at two different positions within the short repeat:

```
12345
```

```
        TTCTC
POS=1   T--TC
POS=3   TTC--
```

In order to be able to compare indels between two datasets, we left-align such variants.

Q5: Use the bcftools norm command to normalize the filtered callset. Note that you will need to provide the `--fasta-ref` option. Check in the output how many indels were realigned.

[ ]:

Now continue to the next section of the tutorial: Multi-sample variant calling

4 Calling Variants Across Multiple Samples

In many types of experiments we sequence multiple samples and compare their genetic variation across samples. The single-sample variant calling we have done so far has the disadvantage of not providing information about reference genotypes. Because only variant sites are stored, we are not able to distinguish between records missing due to reference genotypes versus records missing due to lack of coverage.

In this section we will call variants across two mouse samples.

To begin, check that there are two BAM files in the directory.

[ ]:
```
ls *.bam
```

Now modify the variant calling command from the previous section to use both BAM files. Write the output to a BCF file called `multi.bcf`.

[ ]:

Now index the file `multi.bcf`

[ ]:

Filter the file `multi.bcf` using the same filters as the previous section and write the output to a BCF file called `multi.filt.bcf`.

[ ]:

Now index the `multi.filt.bcf` file.

[ ]:

4.1 Exercises

Q1: What is the ts/tv of the raw calls and of the filtered set?

Q2: What is the ts/tv of the removed sites?

Now continue to the next section of the tutorial: Visualising variants

5 Variant visualisation

It is often useful to visually inspect a SNP or indel of interest in order to assess the quality of the variant and interpret the genomic context of the variant. We can use the IGV tool to view some of the variant positions from the VCF file.

Start IGV by typing:

[ ]:
```
igv
```

5.1 Load the reference genome

Open the mouse reference genome"

Go to ' *Genomes -> Select Hosted Genome* ' and select "Mouse mm10". This is a synonym for GRCm38, which is the current mouse assembly (reference genome)

5.2 Load the alignment

Load the alignment file for the sample A_J (A_J.bam). Before loading, make sure the the file is indexed (`samtools index A_J.bam`).

Go to ' *File -> Load from File...* '. Select the "A_J.bam" BAM file that you created in the previous section and click' *Open* '.

5.3 Exercises

Use the IGV navigation bar, go to the region chr19:10,001,874-10,002,017 and inspect the SNP at position 10001946.

Q1: How many forward aligned reads support the SNP call?

Hint Hover the mouse pointer over the coverage bar at the top (or click, depending on the IGV settings) to get this information.

Q2: Was this SNP called by bcftools?

Hint Use `bcftools view -H -r 19:10001946 multi.filt.bcf` to verify

Q3: Did this SNP pass the filters?

Hint Look for this information in the BCF file

Use the IGV navigation bar, go to the region chr19:10072443 and inspect the SNP at position 10072443.

Q4: Was this SNP called by bcftools?

Q5: Did the SNP pass the filters?

Q6: Does this look like a real SNP? Please explain why.

Now continue to the next section of the tutorial: Variant annotation

6 Variant annotation

Variant annotation is used to help researchers filter and prioritise functionally important variants for further study. There are several popular programs available for annotating variants. These include:

- bcftools csq
- Ensembl VEP (Variant Effect Predictor)
- SnpEff

These tools can be used to to predict the functional consequence of the variants on the protien (e.g. whether a variant is mis-sense, stop-gain, frameshift inducing etc).

6.1 bcftools csq

Here we will use the lightweight `bcftools csq` command to annotate the variants. Type the command:

[ ]:
```
bcftools view -i 'FILTER="PASS"' multi.filt.bcf | bcftools csq -p m -f␣
↪GRCm38_68.19.fa -g Mus_musculus.part.gff3.gz -Ob -o multi.filt.annot.bcf
```

The command takes VCF as input, the `-f` option specifies the reference file that the data was aligned too and the `-g` option specifies the GFF file that contains the gene models for the reference. Because our data is not phased, we provide the `-p` option (which does not actually phase the data, but tells the program to make an assumption about the phase). The `-Ob` option ensures the command produces compressed BCF as output.

Now index the BCF file:

[ ]:
```
bcftools index multi.filt.annot.bcf
```

6.1.1 Exercises

Q1 Use the `bcftools query -f '%BCSQ\n'` command to extract the consequence at position 19:10088937.

[ ]:
```

```

Q2 What is the functional annotation at this site?

[ ]:
```

```

Q3 What is the amino acid change?

[ ]:
```

```

Congratulations you have reached the end of the variant calling tutorial. For the answers to the exercises in this tutorial please visit answers.