Published in final edited form as:
Curr Protoc Bioinformatics. ; 11(1110): 11.10.1–11.10.33. doi:10.1002/0471250953.bi1110s43.

From FastQ data to high confidence variant calls: the Genome Analysis Toolkit best practices pipeline

Geraldine A. Van der Auwera, **Mauricio O. Carneiro**, **Chris Hartl**, **Ryan Poplin**, **Guillermo del Angel**, **Ami Levy-Moonshine**, **Tadeusz Jordan**, **Khalid Shakir**, **David Roazen**, **Joel Thibault**, **Eric Banks**, **Kiran V. Garimella**[1], **David Altshuler**, **Stacey Gabriel**, and **Mark A. DePristo**

Geraldine A. Van der Auwera: vdauwera@broadinstitute.org; Mauricio O. Carneiro: carneiro@broadinstitute.org; Chris Hartl: chartl@broadinstitute.org; Ryan Poplin: rpoplin@broadinstitute.org; Guillermo del Angel: delangel@broadinstitute.org; Ami Levy-Moonshine: ami@broadinstitute.org; Tadeusz Jordan: tjordan@broadinstitute.org; Khalid Shakir: kshakir@broadinstitute.org; David Roazen: droazen@broadinstitute.org; Joel Thibault: thibault@broadinstitute.org; Eric Banks: ebanks@broadinstitute.org; Kiran V. Garimella: kiran.garimella@gmail.com; David Altshuler: altshuler@broadinstitute.org; Stacey Gabriel: stacey@broadinstitute.org; Mark A. DePristo: depristo@broadinstitute.org

Broad Institute, Genome Sequencing and Analysis Group, 301 Binney Street, Cambridge MA 02142

[1]Wellcome Trust Centre for Human Genetics, University of Oxford, Roosevelt Drive, Oxford OX3 7BN, UK

Abstract

This unit describes how to use BWA and the Genome Analysis Toolkit (GATK) to map genome sequencing data to a reference and produce high-quality variant calls that can be used in downstream analyses. The complete workflow includes the core NGS data processing steps that are necessary to make the raw data suitable for analysis by the GATK, as well as the key methods involved in variant discovery using the GATK.

Keywords

NGS; WGS; exome; variant detection; genotyping

INTRODUCTION

Genomic variant discovery seems like a fairly simple problem in principle – take reads mapped to a reference sequence, and at every position, count the mismatches and deduce the genotype. In practice it is made more complicated by multiple sources of errors: amplification biases when the sample libraries are prepped in the wet lab, machine errors when the libraries are sequenced, software errors and mapping artifacts when the reads are aligned. A good variant calling workflow must involve data preparation methods that correct or compensate for these various error modes. Variant discovery can then be performed on the appropriately processed data, using a robust calling algorithm that leverages meta-information such as base qualities. At this stage, it is preferable to prioritize sensitivity (seeking to include as many potential variants as possible to avoid missing any) over specificity (seeking to limit the proportion of false positives in the call set). Once a highly-sensitive call set has been generated, appropriate filters can be applied to achieve the desired balance between sensitivity and specificity.

BWA (Li & Durbin, 2009) and GATK (DePristo et al., 2011; McKenna et al., 2010) are publicly available software packages that can be used to construct a variant calling workflow following those principles. Specifically, BWA is a robust read mapping algorithm that can be used to map sequence data to a reference genome. Its output can be used with minimal processing by utility software packages (Picard and SAMtools) as input to the GATK tools, which make up the rest of the workflow. The GATK, or Genome Analysis Toolkit, was originally developed as a programming framework that allowed easy development of new genome analysis tools. It has since matured into a suite of tools that can be used "out of the box" to perform workflows such as the one detailed in this unit.

The examples given in this unit follow the Best Practices recommendations formulated by the GATK development team, the Genome Sequencing and Analysis (GSA) group at the Broad Institute. These recommendations have been developed on the basis of extensive experimentation, largely in the context of the 1000 Genomes Project, and they continue to evolve as the technologies improve. Here, they are applied to a simple but real dataset consisting of whole genome sequence (WGS) data generated from the GSA group's favorite test sample, NA12878, on an Illumina HiSeq machine. Most of the examples will focus on chromosome 20 in order to keep runtimes down, but all commands can be extended to full genome analysis very simply, as detailed in the text. An overview of the workflow is given in the Strategic Planning section below.

STRATEGIC PLANNING

The workflow presented in this unit can be divided into three main sections that are meant to be performed sequentially, as illustrated in the accompanying Figure 1. For each section, the Basic Protocol is the default protocol you should follow when working on your own data. Alternate Protocols are provided to show how you should adapt your workflow when you are working with datasets that do not fulfill the standard assumptions that are made in the Basic Protocols. Support Protocols are provided to show how to deal with common issues that arise when using these tools.

Support Protocol 1 describes how to obtain, install and test all of the software packages required by the protocols in this unit. You should complete this protocol before attempting to proceed with any of the following. Please make sure also that you read the software descriptions provided by their respective authors, as you will find it very useful to understand what is the purpose of each package.

FASTQ TO BAM (Basic Protocol 1)

In this section, you will learn about the core processing steps that are necessary to transform raw sequencing data into a BAM file that is suitable for analysis. These steps include:

1. Mapping the reads to a reference (BWA)
2. Formatting the data for input to the GATK (Picard)

3. Performing local indel realignment (GATK)
4. Performing base quality score recalibration (GATK)
5. Performing data compression (GATK)

Although we do not provide an Alternate Protocol for this section, there are various possible deviations from Basic Protocol 1 at the alignment step. You may use other aligners than BWA, either by choice or out of necessity (e.g. if you are working with RNAseq data, in which case you need to use an aligner specifically developed for that purpose such as Bowtie; UNIT 11.7). We recommend using BWA because we have found that it gives the best results for the types of data that we deal with most regularly, and it conveniently outputs a data format that Picard tools can read natively without additional manipulations.

Please note also that the example we used to illustrate the application of this protocol is an extremely simple one. Here, you will be working with whole-genome sequence (WGS) of a single sample, sequenced from one library run on a single sequencing lane of an Illumina-type machine. In practice, you may find yourself dealing with multiple samples, multiples libraries and multiple lanes, arranged in various configurations, which may focus on specific genome regions (e.g. targeted or exome sequencing). A complete guide on how to tailor this workflow to the exact configuration of your dataset is beyond the scope of this work, but you can find advice on how to do this in the online GATK documentation.

In addition to the Basic Protocol, we have provided two Support Protocols that explain how to deal with input datasets that have already been partially processed:

- **Support Protocol 2** details how to revert a BAM file to FASTQ format if you want to discard all the processing that has already been done.
- **Support Protocol 3** details how to fix the formatting of a BAM file if you need to make an existing BAM file compliant with the GATK's requirements (sorting, duplicate marking and read groups).

CALLING VARIANTS (Basic Protocol 2 OR Alternate Protocol 1)

In this section, you will learn how to call variants using either one of the GATK's two different variant callers, the UnifiedGenotyper (UG) and the HaplotypeCaller (HC). The HaplotypeCaller is a more recent and sophisticated tool than the UnifiedGenotyper, and we recommend using HaplotypeCaller in all cases, with only a few exceptions: if you want to analyze more than 100 samples at a time (for performance reasons), if you are working with non-diploid organisms (UG can handle different levels of ploidy while HC cannot) or if you are working with pooled samples (also due to the HC's limitation regarding ploidy).

FILTERING VARIANTS (Basic Protocol 3 OR Alternate Protocol 2)

In this section, you will learn how to filter the raw calls emitted by the GATK callers using either one of two methods, variant quality score recalibration method and hard-filtering. The variant quality score recalibration method uses machine learning to filter variants and is capable of producing call sets fine-tuned to a desired level of truth sensitivity. It is the method we recommend by default because it is more powerful and less bias-prone than the hard-filtering method, but due to the limitations of the underlying statistical model, it can only be used on datasets that include a minimum number of variants. For smaller datasets (also in most non-human datasets) the hard-filtering method must be used as an alternative.

ANNOTATING VARIANTS (Support Protocol 4)

Variant annotations play a major role in downstream analyses, and they are also extensively used for filtering call sets. These annotations are primarily made when the variant calling is performed, but you will often find yourself needing to add variant annotations to a VCF file after the fact, whether before or after the filtering step. For this purpose, we have provided a Support Protocol that explains how to add variant annotations to your call set without having to repeat the variant calling step. You can use Support Protocol 4 to add annotations to any valid VCF call set. Note that most annotations will require that you provide the program with a BAM file containing the sequence data from which the calls were derived.

ADDITIONAL REMARKS REGARDING COMMAND-LINE ARGUMENTS AND PARAMETERS

As mentioned earlier, the examples we have provided to illustrate this unit use a specific data type. Some of the command-line arguments and parameters are specifically adapted for the example data, so when you move on to working with your own datasets, you should always check that you are using the most appropriate parameters. You will find relevant information on how to do this in the online GATK documentation.

In addition, you should be aware that the GATK tools are under ongoing active development. We frequently update the tools with algorithm improvements and expanded functionalities. As a consequence, our recommendations regarding arguments and parameters tend to evolve over time. We urge you to subscribe to notifications on the GATK support forum and frequently check back to the online GATK documentation to stay up-to-date on what are the latest recommendations.

FURTHER ANALYSIS WORK

Once you have completed the variant filtering step, your call set is ready to use in whatever downstream analyses your research requires. The GATK contains various other tools that can be used for such analyses; please see the online GATK documentation for full details. You may also use any third-party software that accepts valid VCF files as input.

BASIC PROTOCOL 1

FROM FASTQ TO ANALYSIS-READY BAM: PREPARING THE SEQUENCE DATA

When you receive your sequence data from your sequencing provider (whether it is an in-house service or a commercial company), the data is typically in a raw state (one or several FASTQ files) that is not suitable for immediate analysis using the GATK. This protocol will walk you through a series of processing steps that are necessary in order to prepare your data for analysis, starting with FASTQ files. The end product of this protocol will be a BAM file that can be used to call variants.

In this protocol series, we are working with data sequenced using one library for one sample, sequenced on one machine lane. For details on how to adapt this protocol to more complex scenarios (multiple libraries, lanes, samples), please see the Best Practices section of the GATK documentation website.

Necessary Resources

- Software

  Several packages are necessary. See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  – Burrows-Wheeler Alignment Tool (BWA)

  – SAMtools

  – Picard Tools

  – Genome Analysis Toolkit (GATK)

  – R and R libraries ggplot2 and gsalib

- Hardware

  A computer with as much memory and computing power as possible. A minimum of 2 Gb of memory is required, but 4 to 8 Gb is recommended if you are working with a genome as large as the human's. The software can run natively on any Unix or Unix-like system such as Linux or MacOS X with Java 1.7 installed. BWA has also been ported to Windows, and users have reported being able to use the GATK and Picard tools on Windows with Cygwin, but this usage is not officially supported.

- Files

  You can find the following example files in the resource bundle associated with this unit located at http://www.broadinstitute.org/.....

  – The data to analyze: sequence reads in FASTQ format (raw_reads.fq)

  – The human reference genome in FASTA format (reference.fa)

  – A database of known variants in VCF format (dbsnp137.vcf)

    Note that all files must be able to pass strict validation of their respective format specifications.

Preparing the reference sequence—The GATK uses two files to safely access the reference file: a dictionary of the contig names and sizes, and an index file to allow efficient random access to the reference bases. You have to generate these files up front in order to be able to use a FASTA file as reference with GATK tools.

1 Generate the BWA index by running the following BWA command:

```
bwa index reference.fa
```

This creates a collection of files used by BWA to perform the alignment. Among these files, there will be a file called “reference.fa.fai”, with one record per line for each of the contigs in the FASTA reference file. Each record is composed of the contig name, size, location, basesPerLine and bytesPerLine.

Having the index readily available allows the GATK to compute exactly where a particular reference base specified by the location “contig:pos” is in the FASTA file, which speeds up processing of large datasets significantly.

2 Generate the fasta file index by running the following SAMtools command:

```
samtools faidx reference.fa
```

This creates a file called “reference.fa.fai”, with one record per line for each of the contigs in the FASTA reference file. Each record is composed of the contig name, size, location, basesPerLine and bytesPerLine.

3 Generate the sequence dictionary by running the following Picard command:

```
java -jar CreateSequenceDictionary.jar \
REFERENCE=reference.fa \
OUTPUT=reference.dict
```

This creates a file called “reference.dict” formatted like a SAM header, describing the contents of your reference FASTA file.

You may notice that the index and dictionary files are not explicitly passed in the GATK commands that you will encounter later in this protocol. That is because the GATK automatically looks for them when you pass in the reference, and it “guesses” their names following an established convention based on the reference name. For this reason, you should not change the names of these two files – or if you do, you should update the reference file name accordingly, and vice versa.

4 Preparing the appropriate read group information

The read group information is where you enter the meta-data about your sample. This line is very important to all downstream analysis since it is the only meta information that will be visible to analysis tools. There are multiple fields in the Read Group tag, but some of them are of criticial importance. In the example below we highlight a minimalistic read group tag with the most important information necessary. We will use this line as a command line parameter for the BWA aligner in the next step.

```
@RG\tID:group1\tSM:sample1\tPL:illumina\tLB:lib1\tPU:unit1
```

where the ’\t’ stands for the tab character

The @RG identifies that this is the “read group” tag. Following the tabs we have the following keys:

- ID – globally unique string identifying this run. In general we use an identifier linked to the lane where the data was run (for illumina data) or instrument where lane is not applicable.
- SM – the name associated with the DNA sample in this file. In general this will be a sample identifier such as NA12878, but it can be any string. This is the most important field for the GATK because all analysis is done by sample, and this is what the tools will use to define what data belongs to which sample.
- PL – the platform used. The list of supported platforms can be found in the GATK’s online manual as it is ever growing. For example you can use “illumina”, “pacbio” or “iontorrent” here.
- LB – an identifier of the library from which this DNA was sequenced. This is important for future reference and quality control. In case errors are associated with the DNA prep, this will be the field linking the data to the laboratory process.
- PU – the platform unit identifier for this run. Any generic identifier that will allow any investigation to go back to the very machine, and time where this data was run. We typically use the “flowcell-barcode.lane” unique identifier for Illumina runs.

  The read group information is key for downstream GATK functionality. The GATK will not work without a read group tag. Make sure to enter as much metadata as you know about your data in the read group fields provided. For more information about all the possible fields in the @RG tag, take a look at the SAM specification (http://samtools.sourceforge.net/SAM1.pdf).

Mapping the data to the reference—Generate a SAM file containing aligned reads by running the following BWA command:

```
bwa mem -R ’<read group info>’ -p reference.fa raw_reads.fq >
aligned_reads.sam
```

In this command, replace read group info by the read group identifier composed in the previous step.

This creates a file called “aligned_reads.sam” containing the aligned reads from all input files, combined, annotated and aligned to the same reference.

Here we are using a command that is specific for pair ended data in an interleaved fastq file, which is what we are providing to you as a tutorial file. To map other types of datasets (e.g. single-ended or pair-ended in forward/reverse read files) you will need to adapt the command accordingly. Please see the BWA documentation for exact usage and more options for these commands.

Converting to BAM, sorting and marking duplicates—The next set of pre-processing operations format the data to suit the requirements of the GATK tools by converting the mapping data to a sorted de-duplicated BAM file.

5 Perform all three operations simultaneously by running the following Picard command:

```
java -jar MarkDuplicates.jar \
INPUT=aligned_reads.sam \
OUTPUT=dedup_reads.bam \
SO=coordinate
```

This creates a sorted BAM file called “dedup_reads.bam” with the same content as the input file, except that any duplicate reads are marked as such.

During the sequencing process, the same DNA molecules can be sequenced several times. The resulting duplicate reads are not informative and should not be counted as additional evidence for or against a putative variant. The duplicate marking process (sometimes called “dedupping” in bioinformatics slang) identifies these reads as such so that the GATK tools know to ignore them.

Is all of this really necessary? The GATK is notorious for imposing strict formatting guidelines and requiring the presence of information such as read groups that other software packages do not require. Although this represents a small additional processing burden upfront, the downstream benefits are numerous, including the ability to process library data individually, and significant gains in speed and parallelization options.

Local realignment around indels—The mapping algorithms that are used in the initial step of aligning the data to the reference are prone to various types of artifacts. For example, reads that align on the edges of indels often get mapped with mismatching bases that might look like evidence for SNPs, but are actually mapping artifacts. The realignment process identifies the most consistent placement of the reads with respect to the indel in order to clean up these artifacts. It occurs in two steps: first the program identifies intervals that need to be realigned, then in the second step it determines the optimal consensus sequence and performs the actual realignment of reads.

6 Create a target list of intervals to be realigned by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T RealignerTargetCreator \
-R reference.fa \
-I dedup_reads.bam \
-L 20 \
-known gold_indels.vcf \
-o target_intervals.list
```

This creates a file called "target_intervals.list" containing the list of intervals that the program identified as needing realignment within our target, chromosome 20.

The list of known indel sites (gold_indels) are used as targets for realignment. Only use it if there is such a list for your organism.

So this way of calling the program and selecting which tool to run is a little like a hybrid of how we called BWA and how we called Picard tools. To put it another way, if BWA is a standalone game device that comes preloaded with several games, Picard tools are individual game cartridges that plug into the Java console, and GATK is a single cartridge that also plugs into the Java console but contains many games.

7 Perform realignment by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T IndelRealigner \
-R reference.fa \
-I dedup_reads.bam \
-targetIntervals target_intervals.list \
-known gold_indels.vcf \
-o realigned_reads.bam
```

This creates a file called "realigned_reads.bam" containing all the original reads, but with better local alignments in the regions that were realigned.

Note that we didn't include the -L 20 argument this time. Not only is it unnecessary, since our realignment targets were all created within chromosome 20, but it would be a problem if we included it. The reason for this is that when given multiple interval definitions, the GATK merges any overlapping intervals together. Since all the target intervals are within chromosome 20, the program would end up trying to unnecessarily realign the entire chromosome.

Base quality score recalibration (BQSR)—The per-base estimate of error known as the base quality score is the foundation upon which all statistically calling algorithms are based. We have found that the estimates provided by the sequencing machines are often inaccurate and/or biased. The recalibration process applies an empirically accurate error model to the bases, producing a BAM file that is suitable for analysis.

8 Analyze patterns of covariation in the sequence dataset by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T BaseRecalibrator \
-R reference.fa \
-I realigned_reads.bam \
-L 20 \
-knownSites dbsnp.vcf \
-knownSites gold_indels.vcf \
-o recal_data.grp \
-plots before_recal.pdf
```

This creates a GATKReport file called “recal_data.grp” containing several tables. These tables contain the covariation data that will be used in a later step to recalibrate the base qualities of your sequence data. This also generates a plot called “before_recal.pdf” showing how the original base qualities in your data differ from the base qualities empirically calculated by the BaseRecalibrator.

It is imperative that you provide the program with a set of known sites, otherwise it will refuse to run. The known sites are used to build the covariation model and estimate empirical base qualities. For details on what to do if there are no known sites available for your organism of study, please see the online GATK documentation.

9 Generate a plot showing how base qualities will look like after recalibration by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T BaseRecalibrator \
-R reference.fa \
-I realigned_reads.bam \
-L 20 \
-BQSR recal_data.grp \
-o post_recal_data.grp \
-plots after_recal.pdf
```

This creates another GATKReport file, but we don’t need this one so you can just ignore it. What’s important here is the new plot, called “after_recal.pdf”, which shows how the recalibrated base qualities will match up to the base qualities empirically calculated by the BaseRecalibrator. Comparing the “before” and “after” plots allows you to check the effect of the base recalibration process before you actually apply the recalibration to your sequence data. This “previewing” is achieved thanks to the **-BQSR** flag, which tells the GATK engine to perform on-the-fly recalibration based on the recalibration data table.

For details on how to interpret the base recalibration plots, please see http://gatkforums.broadinstitute.org/discussion/44/base-quality-score-recalibration-bqsr.

10 Apply the recalibration to your sequence data by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T PrintReads \
-R reference.fa \
-I realigned_reads.bam \
-L 20 \
-BQSR recal_data.grp \
-o recal_reads.bam
```

This creates a file called "recal_reads.bam" containing all the original reads, but now with exquisitely accurate base substitution, insertion and deletion quality scores. By default, the original quality scores are discarded in order to keep the file size down. However, you have the option to retain them by adding the flag **–emit_original_quals** to the PrintReads command, in which case the original qualities will also be written in the file, tagged **OQ**.

Notice how this step uses a very simple tool, **PrintReads**, *to apply the recalibration. What's happening here is that we are loading in the original sequence data, having the GATK engine recalibrate the base qualities on-the-fly thanks to the* **-BQSR** *flag (as explained earlier), and just using PrintReads to write out the resulting data to the new file.*

Data compression with ReduceReads (optional)—ReduceReads is a novel data compression algorithm. Its purpose is to take a BAM file with sequence data and reduce it down to just the information necessary to make accurate SNP and indel calls, as well as genotype reference sites using GATK tools like UnifiedGenotyper or HaplotypeCaller. ReduceReads accepts as an input a BAM file and produces a valid BAM file, but with a few extra tags that the GATK tools can use to make accurate calls.

11 Compress your sequence data by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T ReduceReads \
-R reference.fa \
-I recal_reads.bam \
-L 20 \
-o reduced_reads.bam
```

This creates a file called "reduced_reads.bam" containing only the sequence information that is essential for calling variants.

ReduceReads works well for any high-coverage (at least 20x average coverage) BAM file, but is not recommended for low coverage data (less than 10x average coverage). In this case we highly recommend using ReduceReads to minimize the file sizes. Using ReduceReads on your BAM files will bring down the sizes to approximately 1/100 of their original sizes, allowing the GATK to process tens of thousands of samples simultaneously without excessive IO and processing burdens. Even for single samples ReduceReads cuts the memory requirements, IO burden, and CPU costs of downstream tools significantly (10x or more) and so we recommend you preprocess analysis-ready BAM files with ReduceReads.

Note however that ReduceReads performs a lossy compression of the sequencing data that works well with the downstream GATK tools, but may not be supported by external tools. Therefore, we recommend that you archive your original BAM file, or at least a copy of your original FASTQs, as ReduceReads is highly lossy and does not qualify as an archive data compression format.

BASIC PROTOCOL 2

FROM ANALYSIS-READY BAM TO RAW VARIANTS: CALLING VARIANTS IN DIPLOID ORGANISMS WITH HaplotypeCaller

Now that your data is properly prepared as an analysis-ready BAM file, it is time to "call variants" – that is, identify the sites where your data displays variation relative to the reference genome. In this protocol we will focus on calling two types of variation, single-nucleotide polymorphisms (SNPs) and insertion-deletions (Indels) using the HaplotypeCaller. The end product of this protocol will be a VCF file containing raw calls that should be filtered before they can be used in downstream analyses.

Many variant callers specialize in either SNPs or Indels, or (like the GATK's own UnifiedGenotyper) have to call them using separate models of variation. The HaplotypeCaller is capable of calling SNPs and indels simultaneously via local de-novo assembly of haplotypes in an active region. In other words, whenever the program encounters a region showing signs of variation, it discards the existing mapping information and completely reassembles the reads in that region. This allows the HaplotypeCaller to be more accurate when calling regions that are traditionally difficult to call, for example when they contain different types of variants close to each other. It also makes the HC much better at calling indels.

Necessary Resources

- Hardware

Same as described for Basic Protocol 1.

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  – Genome Analysis Toolkit (GATK)

- Files

  All files must be able to pass strict validation of their respective format specifications.

  – The sequence data in BAM format, processed as described in Basic Protocol 1 (reduced_reads.bam)

  – The human reference genome in FASTA format (reference.fa)

Calling variants with HaplotypeCaller

1 Determine the basic parameters of the analysis. If you do not specify these parameters yourself, the program will use default values. However we recommend that you set them explicitly because it will help you understand how the results are bounded and how you can modify the program's behavior.

- Genotyping mode (–*genotyping_mode*)

  This specifies how we want the program to determine the alternate alleles to use for genotyping. In the default DISCOVERY mode, the program will choose the most likely alleles out of those it sees in the data. In GENOTYPE_GIVEN_ALLELES mode, the program will only use the alleles passed in from a VCF file (using the -*alleles* argument). This is useful if you just want to determine if a sample has a specific genotype of interest and you are not interested in other alleles.

- Output mode (–*output_mode*)

  This option specifies which type of calls we want the program to emit. In the default EMIT_VARIANTS_ONLY mode, the program will emit calls only for sites that appear to be variant (whether with high or low confidence). In EMIT_ALL_CONFIDENT_SITES mode, the program will emit calls at sites where it has high confidence in whether the site is either variant or non-variant (i. e. has the reference allele). In EMIT_ALL_SITES mode, the program will emit calls at any callable site (i. e. site where there is useable data) regardless of confidence. This last argument is intended only for point mutations (SNPs) in DISCOVERY genotyping mode, or generally when running in GENOTYPE_GIVEN_ALLELES genotyping mode. There is no guarantee that it can produce a comprehensive set of indels in DISCOVERY mode.

- Emission confidence threshold (–*stand_emit_conf*)

This is the minimum confidence threshold (phred-scaled) at which the program should emit sites that appear to be possibly variant.

- Calling confidence threshold (*–stand_call_conf)*

  This is the minimum confidence threshold (phred-scaled) at which the program should emit variant sites as called. If a site's associated genotype has a confidence score lower than the calling threshold, the program will emit the site as filtered and will annotate it as LowQual. This threshold separates high confidence calls from low confidence calls.

  The terms **called** *and* **filtered** *are tricky because they can mean different things depending on context. In ordinary language, people often say a site was called if it was emitted as* **a** *variant. But in the GATK's technical language, saying a site was called means that that site passed the confidence threshold test. For filtered, it's even more confusing, because in ordinary language, when people say that sites were filtered, they usually mean that those sites successfully passed a filtering test. However, in the GATK's technical language, the same phrase (saying that sites were filtered) means that those sites failed the filtering test. In effect, it means that those would be filtered out if the filter was used to actually remove low-confidence calls from the callset, instead of just tagging them. In both cases, both usages are valid depending on the point of view of the person who is reporting the results. So it's always important to check what is the context when interpreting results that include these terms.*

2 Call variants in your sequence data by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T HaplotypeCaller \
-R reference.fa \
-I reduced_reads.bam \
-L 20 \
--genotyping_mode DISCOVERY \
--output_mode EMIT_VARIANTS_ONLY \
--stand_emit_conf 10 \
--stand_call_conf 30 \
-o raw_variants.vcf
```

This creates a VCF file called raw_variants.vcf, containing all the sites that the HaplotypeCaller evaluated to be potentially variant. Note that this file contains both SNPs and Indels.

Although you now have a nice fresh set of variant calls, the variant discovery stage is not over. The distinctions made by the caller itself between low-confidence calls and the rest are very primitive, and should not be taken as a definitive guide for filtering. The GATK callers are designed to be very lenient in calling variants, so it is extremely important to apply one of the filtering methods detailed in Basic Protocol 3 and Alternate Protocol 2, in order to move on to downstream analyses with the highest-quality call set possible.

BASIC PROTOCOL 3

FROM RAW TO ANALYSIS-READY VARIANTS: VARIANT QUALITY SCORE RECALIBRATION

The GATK callers (HaplotypeCaller and UnifiedGenotyper) are by design very lenient in calling variants in order to achieve a high degree of sensitivity. This is a good thing because it minimizes the chance of missing real variants, but it does mean that we need to refine the call set to reduce the amount of false positives, which can be quite large. The best way to perform this refinement is to use variant quality score recalibration (VQSR). In the first step of this two-step process, the program uses machine learning methods to assign a well-calibrated probability to each variant call in a raw call set. We can then use this variant quality score in the second step to filter the raw call set, thus producing a subset of calls with our desired level of quality, fine-tuned to balance specificity and sensitivity.

In this protocol, we show you how to perform VQSR on the raw call set you generated with the HaplotypeCaller using Basic Protocol 2. The end product of this protocol will be a VCF file containing high-quality variant calls that can be used in downstream analyses.

To calculate the variant quality score, the program builds an adaptive error model using training sets (explained further below). The model provides a continuous, covarying estimate of the relationship between variant call annotations and the probability that a variant call is a true genetic variant, rather than a sequencing or data processing artifact. The program then applies this adaptive error model to both known and novel variation discovered in the call set of interest, and annotates each variant with a quality score called VQSLOD. This is the log odds ratio of the variant call being a true positive versus being a false positive according to the training model.

Necessary Resources

- Hardware

  Same as described for Basic Protocol 1.

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  – Genome Analysis Toolkit (GATK)

  – RStudio and the R libraries ggplot2 and gsalib

- Files

All files must be able to pass strict validation of their respective format specifications.

- Call set in VCF format, produced as described in Basic Protocol 2 (raw_variants.vcf)
- The human reference genome in FASTA format (reference.fa)
- Sets of known/true variants in VCF format for training the model:
    - ♦ HapMap 3.3 sites (hapmap.vcf)
    - ♦ Omni 2.5 sites (omni.vcf)
    - ♦ 1000G high-confidence sites (1000G.vcf)
    - ♦ dbSNP (dbsnp.vcf)
    - ♦ Mills & 1000G Gold Standard Indels (mills.vcf)

We have found that SNPs and Indels, being different classes of variation, can have different “signatures” that indicate whether they are real or artifactual. For that reason, we will run the the variant recalibration process separately on SNPs and Indels, in a sequential manner. This allows the program to build separate error models for each, ensuring that their specific signatures don’t get mixed up. Conveniently, the variant recalibration tools are capable of analyzing and recalibrating one type of variation without affecting the other, so we don’t need to separate the variants into different files.

Recalibrating variant quality scores for SNPs

3 Specify which call sets the program should use as resources to build the recalibration model. For each training set, we use key-value tags to qualify whether the set contains known sites, training sites, and/or truth sites. We also use a tag to specify the prior likelihood that those sites are true (using the Phred scale).

- True sites training resource: HapMap (International HapMap 3 Consortium et al., 2010)

  This resource is a SNP call set that has been validated to a very high degree of confidence. The program will consider that the variants in this resource are representative of true sites (*truth=true*), and will use them to train the recalibration model (*training=true*). We will also use these sites later on to choose a threshold for filtering variants based on sensitivity to truth sites. The prior likelihood we assign to these variants is Q15 (96.84%).

- True sites training resource: Omni

  This resource is a set of polymorphic SNP sites produced by the 1000 Genomes project Omni genotyping array(Durbin et al., 2010). The program will consider that the variants in this resource are representative of true sites (*truth=true*), and will use them to train the recalibration model (*training=true*). The prior likelihood we assign to these variants is Q12 (93.69%).

- Non-true sites training resource: 1000G(Durbin et al., 2010)

  This resource is a set of high-confidence SNP sites produced by the 1000 Genomes Project. The program will consider that the variants in this resource may contain true variants as well as false positives (*truth=false*), and will use them to train the recalibration model (*training=true*). The prior likelihood we assign to these variants is Q10.

- Known sites resource, not used in training: dbSNP(Sherry et al., 2001)

  This resource is a SNP call set that has **not** been validated to a high degree of confidence (*truth=false*). The program will **not** use the variants in this resource to train the recalibration model (*training=false*). However, the program will use these to stratify output metrics such as Ti/Tv ratio by whether variants are present in dbsnp or not (*known=true*). The prior likelihood we assign to these variants is Q2 (36.90%).

  The default prior likelihood assigned to all other variants is Q2 (36.90%). This low value reflects the fact that the philosophy of the GATK callers is to produce a large, highly sensitive callset that needs to be heavily refined through additional filtering.

4 Specify which annotations the program should use to evaluate the likelihood of SNPs being real. These annotations are included in the information generated for each variant call by the caller. If an annotation is missing (typically because it was omitted from the calling command) it can be added using the VariantAnnotator tool as described in Support Protocol 4.

- Coverage (DP)

  Total (unfiltered) depth of coverage.

- QualByDepth (QD)

  Variant confidence (from the QUAL field)/unfiltered depth of non-reference samples.

- FisherStrand (FS)

  Phred-scaled p-value using Fisher’s Exact Test(Fisher, 1922) to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- MappingQualityRankSumTest (MQRankSum)

  The u-based z-approximation from the Mann-Whitney Rank Sum Test(Mann and Whitney, 1947) for mapping qualities (reads with ref bases vs. those with the alternate allele). Note that the mapping quality rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

- ReadPosRankSumTest (ReadPosRankSum)

  The u-based z-approximation from the Mann-Whitney Rank Sum Test(Mann and Whitney, 1947) for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

5 Specify the desired truth sensitivity threshold values that the program should use to generate tranches.

- First tranche threshold

  100.0

- Second tranche threshold

  99.9

- Third tranche threshold

  99.0

- Fourth tranche threshold

  90.0

  Tranches are essentially slices of variants, ranked by VQSLOD, bounded by the threshold values specified in this step. The threshold values themselves refer to the sensitivity we can obtain when we apply them to the call sets that the program uses to train the model. The idea is that the lowest tranche is highly specific but less sensitive (there are very few false positives but potentially many false negatives, i.e. missing calls), and each subsequent tranche in turn introduces additional true positive calls along with a growing number of false positive calls. This allows us to filter variants based on how sensitive we want the call set to be, rather than applying hard filters and then only evaluating how sensitive the call set is using post hoc methods.

6 Determine additional model parameters.

- Percentage of bad variants (-*percentBad*)

  0.01

  This is the percentage of the worst scoring variants to use when building the model of bad variants. The 0.01 value means “take the bottom 1 percent”.

- Minimum number of bad variants (-*minNumBad*)

1000

This is the minimum number of worst scoring variants to use when building the model of bad variants. This will override the *-percentBad* argument if necessary, i.e. if the percentage specified does not amount to enough variants as specified here.

7 Build the SNP recalibration model by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T VariantRecalibrator \
-R reference.fa \
-input raw_variants.vcf \
-resource:hapmap,known=false,training=true,truth=true,prior=15.0
hapmap.vcf \
-resource:omni,known=false,training=true,truth=false,prior=12.0
omni.vcf \
-resource:1000G,known=false,training=true,truth=false,prior=10.0
1000G.vcf \
-resource:dbsnp,known=true,training=false,truth=false,prior=2.0
dbsnp.vcf \
-an DP \
-an QD \
-an FS \
-an MQRankSum \
-an ReadPosRankSum \
-mode SNP \
-tranche [100.0, 99.9, 99.0, 90.0] \
-percentBad 0.01 \
-minNumBad 1000 \
-recalFile recalibrate_SNP.recal \
-tranchesFile recalibrate_SNP.tranches \
-rscriptFile recalibrate_SNP_plots.R
```

This creates several files. The most important file is the recalibration report, called “recalibrate_SNP.recal”, which contains the recalibration data. This is what the program will use in the next step to generate a VCF file in which the variants are annotated with their recalibrated quality scores. There is also a file called “recalibrate_SNP.tranches”, which contains the quality score thresholds corresponding to the tranches specified in the original command. Finally, if your installation of R and the libraries listed in Support Protocol 1 was done correctly, you will also find some PDF files containing plots. These plots illustrated the distribution of variants according to certain dimensions of the model.

For detailed instructions on how to interpret these plots, please refer to the online GATK documentation.

8 Apply the desired level of recalibration to the SNPs in the call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T ApplyRecalibration \
-R reference.fa \
-input raw_variants.vcf \
-mode SNP \
--ts_filter_level 99.0 \
-recalFile recalibrate_SNP.recal \
-tranchesFile recalibrate_SNP.tranches \
-o recalibrated_snps_raw_indels.vcf
```

This creates a new VCF file, called "recalibrated_snps_raw_indels.vcf", which contains all the original variants from the original "raw_variants.vcf" file, but now the SNPs are annotated with their recalibrated quality scores (VQSLOD) and either **PASS** or **FILTER** depending on whether or not they are included in the selected tranche.

Here we are taking the second lowest of the tranches specified in the original recalibration command. This means that we are applying to our data set the level of sensitivity that would allow us to retrieve 99% of true variants from the truth training sets of HapMap and Omni SNPs. If we wanted to be more specific (and therefore have less risk of including false positives, at the risk of missing real sites) we could take the very lowest tranche, which would only retrieve 90% of the truth training sites. If we wanted to be more sensitive (and therefore less specific, at the risk of including more false positives) we could take the higher tranches. In our Best Practices documentation, we recommend taking the second highest tranche (99.9%) which provides the highest sensitivity you can get while still being acceptably specific.

Recalibrating variant quality scores for Indels

9 Specify which call sets the program should use as resources to build the recalibration model. For each training set, we use key-value tags to qualify whether the set contains known sites, training sites, and/or truth sites. We also use a tag to specify the prior likelihood that those sites are true (using the Phred scale).

- Known and true sites training resource: Mills indel dataset(Mills et al., 2006)

  This resource is an Indel call set that has been validated to a high degree of confidence. The program will consider that the variants in this resource are representative of true sites (*truth=true*), and will use them to train the recalibration model (*training=true*). The prior likelihood we assign to these variants is Q12 (93.69%).

The default prior likelihood assigned to all other variants is Q2 (36.90%). This low value reflects the fact that the philosophy of the GATK callers is to produce a large, highly sensitive callset that needs to be heavily refined through additional filtering.

10 Repeat steps 4–5, but now choose the annotations to be used to distinguish real **Indels** from artifacts of the data. Any missing annotation can be added using VariantAnnotator as described in step 4.

11 Determine additional model parameters.

- Percentage of bad variants (-*percentBad*)

  0.01

  This is the percentage of the worst scoring variants that the program should use when building the model of bad variants. The 0.01 value means “take the bottom 1 percent”.

- Minimum number of bad variants (-*minNumBad*)

  1000

  This is the minimum number of worst scoring variants that the program should use when building the model of bad variants. This will override the -*percentBad* argument if necessary, i.e. if the percentage specified does not amount to enough variants as specified here.

- Maximum number of Gaussians (-*maxGaussians*)

  4

  This is the maximum number of Gaussians (i.e. clusters of variants that have similar properties) that the program should try to identify when it runs the variational Bayes algorithm that underlies the machine learning method. In essence, this limits the number of different “profiles” of variants that the program will try to identify. This number should only be increased for datasets that include very many variants.

12 Build the Indel recalibration model by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T VariantRecalibrator \
-R reference.fa \
-input recalibrated_snps_raw_indels.vcf \
-
resource:mills,known=true,training=true,truth=true,prior=12.0
```

```
mills.vcf \
-an DP \
-an FS \
-an MQRankSum \
-an ReadPosRankSum \
-mode INDEL \
-tranche [100.0, 99.9, 99.0, 90.0] \
-percentBad 0.01 \
-minNumBad 1000 \
-maxGaussians 4 \
-recalFile recalibrate_INDEL.recal \
-tranchesFile recalibrate_INDEL.tranches \
-rscriptFile recalibrate_INDEL_plots.R
```

This creates several files. The most important file is the recalibration report, called “recalibrate_INDEL.recal”, which contains the recalibration data. This is what the program will use in the next step to generate a VCF file in which the variants are annotated with their recalibrated quality scores. There is also a file called “recalibrate_INDEL.tranches”, which contains the quality score thresholds corresponding to the tranches specified in the original command. Finally, if your installation of R and the libraries listed in Support Protocol 1 was done correctly, you will also find some PDF files containing plots. These plots illustrated the distribution of variants according to certain dimensions of the model.

For detailed instructions on how to interpret these plots, please refer to the online GATK documentation.

13 Apply the desired level of recalibration to the Indels in the call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T ApplyRecalibration \
-R reference.fa \
-input recalibrated_snps_raw_indels.vcf \
-mode INDEL \
--ts_filter_level 99.0 \
-recalFile recalibrate_INDEL.recal \
-tranchesFile recalibrate_INDEL.tranches \
-o recalibrated_variants.vcf
```

This creates a new VCF file, called “recalibrated_variants.vcf”, which contains all the original variants from the original “recalibrated_snps_raw_indels.vcf” file, but now the Indels are also annotated with their recalibrated quality scores

(VQSLOD) and either **PASS** or **FILTER** depending on whether or not they are included in the selected tranche.

Here we are taking the second lowest of the tranches specified in the original recalibration command. This means that we are applying to our data set the level of sensitivity that would allow us to retrieve 99% of true variants from the truth training sets of HapMap and Omni SNPs. If we wanted to be more specific (and therefore have less risk of including false positives, at the risk of missing real sites) we could take the very lowest tranche, which would only retrieve 90% of the truth training sites. If we wanted to be more sensitive (and therefore less specific, at the risk of including more false positives) we could take the higher tranches. In our Best Practices documentation, we recommend taking the second highest tranche (99.9%) which provides the highest sensitivity you can get while still being acceptably specific.

ALTERNATE PROTOCOL 1 (alternate to BP2)

FROM ANALYSIS-READY BAM TO RAW VARIANTS: CALLING VARIANTS IN NON-DIPLOID ORGANISMS WITH UnifiedGenotyper

Although we generally recommend using the HaplotypeCaller for calling variants, in some cases it is not possible to do so, as explained in the Strategic overview. In those cases – when you're processing a high number of samples, working with non-diploid organisms or with pooled samples – you should use the UnifiedGenotyper instead. In this protocol we illustrate the case of calling variants on a non-diploid organism. The end product of this protocol will be a VCF file containing raw calls that should be filtered before they can be used in downstream analyses.

The Unified Genotyper calls SNPs and indels separately by considering each variant locus independently. The model it uses to do so has been generalized to work with data from organisms of any ploidy.

Necessary Resources

- Hardware

  Same as described for Basic Protocol 1.

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  - Genome Analysis Toolkit (GATK)

- Files

  All files must be able to pass strict validation of their respective format specifications.

  - The sequence data in BAM format (haploid_reads.bam) (the data in this file has already been processed for you as described in Basic Protocol 1)

– The human reference genome in FASTA format (reference.fa)

Calling variants with UnifiedGenotyper

1 Determine the basic parameters of the analysis. If you do not specify these parameters yourself, the program will use default values. However we recommend that you set them explicitly because it will help you understand how the results are bounded and how you can modify the program's behavior.

- Ploidy (-*ploidy*)

  In its basic use, this is the ploidy (number of copies of each chromosomes) per sample. By default it is set to 2, to process diploid organisms' genomes, but it can be set to any other desired value, starting at 1 for haploid organisms, and up for polyploids. This argument can also be used to handle pooled data. For that purpose, you'll need to set **-ploidy** to *Number of samples in each pool * Sample Ploidy*. There is no fixed upper limit, but keep in mind that high-level ploidy will increase processing times since the calculations involved are more complex. For full details on how to process pooled data, see Eran *et al.* (in preparation).

- Genotype likelihood model (-*glm*)

  This is the model that the program will use to calculate the genotype likelihoods. By default, it is set to SNP, but it can also be set to INDEL or BOTH. If set to BOTH, both SNPs and Indels will be called in the same run and be output to the same variants file.

- Emission confidence threshold (–*stand_emit_conf*)

  Same as described for the HaplotypeCaller in Basic Protocol 2.

- Calling confidence threshold (–*stand_call_conf)*

  Same as described for the HaplotypeCaller in Basic Protocol 2.

2 Call variants in your sequence data by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T UnifiedGenotyper \
-R haploid_reference.fa \
-I haploid_reads.bam \
-L 20 \
-ploidy 1
--glm BOTH \
--stand\_call\_conf 30 \
--stand\_emit\_conf 10 \
-o raw_haploid_variants.vcf
```

This creates a VCF file called raw_haploid_variants.vcf, containing all the sites that the UnifiedGenotyper evaluated to be potentially variant.

Although you now have a nice fresh set of variant calls, the variant discovery stage is not over. The distinctions made by the caller itself between low-confidence calls and the rest is very primitive, and should not be taken as a definitive guide for filtering. The GATK callers are designed to be very lenient in calling variants, so it is extremely important to apply one of the filtering methods detailed in Basic Protocol 3 and Alternate Protocol 2, in order to move on to downstream analyses with the highest-quality call set possible.

ALTERNATE PROTOCOL 2 (alternate to BP3)

FROM RAW TO ANALYSIS-READY VARIANTS: HARD-FILTERING SMALL DATASETS

Sometimes, your data set is just too small for variant recalibration to work properly – there are not enough variants for the statistical model-building methods underpinning VQSR to be applicable. Or there are no truth site training resources available for your organism, so you have nothing for the program to use to train the VQSR machine learning model. *Note that there ways to generate training resources yourself using your own data, but those are advanced methods that are beyond the scope of this protocol.* In either case, you'll have to resort to using the method that people used before VQSR came about: hard-filtering using fixed thresholds on specific variant annotations.

The problem here is that there is no magic formula to determine which annotations to filter on and what threshold values to use. It depends a lot on various properties of the data. That's why VQSR is so convenient, since it lets the program learn what are those properties without you having to make too many assumptions! Nevertheless, we have developed some general recommendations based on empirical observations that tend to hold up for most standard datasets. Just keep in mind that it is always a good idea to experiment with these recommendations, tweak the values and try to optimize the filters to fit the properties of your particular dataset.

This protocol will show you how to compose hard-filtering expressions and filter the raw call set you generated with the HaplotypeCaller (using Basic Protocol 2) using VariantFiltration. The end product of this protocol will be a VCF file containing high-quality variant calls that can be used in downstream analyses.

Necessary Resources

- Hardware

  Same as described for Basic Protocol 1.

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  – Genome Analysis Toolkit (GATK)

- Files

  All files must be able to pass strict validation of their respective format specifications.

  – Call set in VCF format, produced as described in Basic Protocol 2 (raw_variants.vcf)

  – The human reference genome in FASTA format (reference.fa)

Splitting the call set into separate files containing SNPs and Indels—We have found that SNPs and Indels, being different classes of variation, can have different “signatures” that indicate whether they are real or artifactual. We therefore strongly recommend running the filtering process separately on SNPs and Indels. Unlike what we did for variant recalibration, it is not possible to apply the VariantFiltration tool selectively to only one class of variants (SNP or Indel), so the first thing we have to do is split them into separate VCF files.

3 Extract the SNPs from the call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T SelectVariants \
-R reference.fa \
-V raw_variants.vcf \
-L 20 \
-selectType SNP \
-o raw_snps.vcf
```

This creates a VCF file called raw_snps.vcf, containing just the SNPs from the original file of raw variants.

4 Extract the Indels from the call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T SelectVariants \
-R reference.fa \
-V raw_HC_variants.vcf \
-L 20 \
-selectType INDEL \
-o raw_indels.vcf
```

This creates a VCF file called raw_indels.vcf, containing just the Indels from the original file of raw variants.

Hard-filtering SNPs

5 Determine parameters for filtering SNPs.

SNPs matching any of these conditions will be considered bad and filtered out, i.e. marked FILTER *in the output VCF file. The program will specify which parameter was chiefly responsible for the exclusion of the SNP using the* culprit *annotation. SNPs that do not match any of these conditions will be considered good and marked* PASS *in the output VCF file.*

- QualByDepth (QD)

  2.0

  This is the variant confidence (from the QUAL field) divided by the unfiltered depth of non-reference samples.

- FisherStrand (FS)

  60.0

  Phred-scaled p-value using Fisher's Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- RMSMappingQuality (MQ)

  40.0

  This is the Root Mean Square of the mapping quality of the reads across all samples.

- HaplotypeScore

  13.0

  This is the consistency of the site with two (and only two) segregating haplotypes. Note that this is not applicable for calls made using the UnifiedGenotyper on non-diploid organisms.

- MappingQualityRankSumTest (MQRankSum)

  12.5

  This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for mapping qualities (reads with ref bases vs. those with the alternate allele). Note that the mapping quality rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

- ReadPosRankSumTest (ReadPosRankSum)

  8.0

  This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

6 Apply the filter to the SNP call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T VariantFiltration \
-R reference.fa \
-V raw_snps.vcf \
--filterExpression “QD < 2.0 || FS > 60.0 || MQ < 40.0 ||
HaplotypeScore > 13.0 || MappingQualityRankSum < -12.5 ||
ReadPosRankSum < -8.0” \
--filterName “my_snp_filter” \
-o filtered_snps.vcf
```

This creates a VCF file called “filtered_snps.vcf”, containing all the original SNPs from the “raw_snps.vcf” file, but now the SNPs are annotated with either PASS or FILTER depending on whether or not they passed the filters.

For SNPs that failed the filter, the variant annotation also includes the name of the filter. That way, if you apply several different filters (simultaneously or sequentially), you can keep track of which filter(s) each SNP failed, and later you can retrieve specific subsets of your calls using the SelectVariants tool. To learn more about composing different types of filtering expressions and retrieving subsets of variants using SelectVariants, please see the online GATK documentation.

Hard-filtering Indels

7 Determine parameters for filtering Indels.

Indels matching any of these conditions will be considered bad and **filtered out**, *i.e. marked* FILTER *in the output VCF file. The program will specify which parameter was chiefly responsible for the exclusion of the indel using the* culprit *annotation. Indels that do not match any of these conditions will be considered good and marked* PASS *in the output VCF file.*

- QualByDepth (QD)

  2.0

  This is the variant confidence (from the QUAL field) divided by the unfiltered depth of non-reference samples.

- FisherStrand (FS)

  200.0

Phred-scaled p-value using Fisher's Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- ReadPosRankSumTest (ReadPosRankSum)

  20.0

  This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

8 Apply the filter to the Indel call set by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T VariantFiltration \
-R reference.fa \
-V raw_indels.vcf \
--filterExpression "QD < 2.0 || FS > 200.0 ||
ReadPosRankSum < -20.0" \
--filterName "my_indel_filter" \
-o filtered_indels.vcf
```

This creates a VCF file called "filtered_indels.vcf", containing all the original Indels from the "raw_indels.vcf" file, but now the Indels are annotated with either PASS or FILTER depending on whether or not they passed the filters.

For Indels that failed the filter, the variant annotation also includes the name of the filter. That way, if you apply several different filters (simultaneously or sequentially), you can keep track of which filter(s) each Indel failed, and later you can retrieve specific subsets of your calls using the SelectVariants tool. To learn more about composing different types of filtering expressions and retrieving subsets of variants using SelectVariants, please see the online GATK documentation.

SUPPORT PROTOCOL 1

A brief description of the software used in the main protocols. This section will go over obtaining and installing all the necessary software for you to successfully complete all the steps in this protocol.

OBTAINING AND INSTALLING THE SOFTWARE USED IN THIS UNIT

Necessary Resources

- Hardware

  Same as described for Basic Protocol 1.

- Software

  See below for detailed instructions on how to obtain and install this software.

  - Web browser
  - ANSI compliant C++ compiler and the tools needed for normal compilations (make, shell, the standard library, tar, gunzip). These tools are usually pre-installed on Linux/Unix systems. On MacOS X, you may need to install the MacOS Xcode tools. See https://developer.apple.com/xcode/ for relevant information and software downloads.
  - Java Runtime Environment version 1.6. All Linux/Unix and MacOS X systems should have a JRE pre-installed, but the version may vary. To test your Java version, run the following command in the shell:

    ```
    java -version
    ```

    This should return a message along the lines of “java version 1.6.0_41” as well as some details on the Runtime Environment (JRE) and Virtual Machine (VM). If you have a version other than 1.6.x, be aware that you may run into trouble with some of the more advanced features of the Picard and GATK tools. The simplest solution is to install an additional JRE and specify which you want to use at the command-line. To find out how to do so, you should seek help from your systems administrator.
  - R language and environment for statistical computing and graphics. See http://www.r-project.org/

- Files

  You will find the following example files in the resource bundle associated with this unit.

  - Sequence reads in FASTQ format (raw_reads.fq)
  - The human reference genome in FASTA format (reference.fa)
  - Example Rscript (testplot.Rscript)
  - Example GATKReport (testplot.grp)

- Additional requirements

  To follow these instructions, you will need to have a basic understanding of the meaning of the following words and command-line operations. If you are unfamiliar with any of the following, you should consult a more experienced colleague or your systems administrator if you have one.

- – Basic Unix environment commands
- – Binary/Executable
- – Compiling a binary
- – Adding a binary to your path
- – Command-line shell, terminal or console
- – Software library

BWA(Li and Durbin, 2010)

1 Read the overview of the BWA software on the project homepage at http://bio-bwa.sourceforge.net/.

2 Download the latest version of the software package from http://sourceforge.net/projects/bio-bwa/files/.

3 Unpack the tar file using:

```
tar xjf bwa-0.7.3a.tar.bz2\
```

This will produce a directory called “bwa-0.7.3a” containing the files necessary to compile the BWA binary.

4 Move to this directory and compile using:

```
cd bwa-0.7.3a
make
```

The compiled binary is called “bwa”. You should find it within the same folder (bwa-0.7.3a in this example).

You may also find other compiled binaries; at time of writing, a second binary called “bwamem-lite” is also included. You can disregard this file for now.

5 Add the BWA binary to your path to make it available on the command line. This completes the installation process.

6 Test your installation of BWA by opening a shell and running:

```
bwa
```

This should print out some version and author information as well as a list of commands. As the “Usage” line states, to use BWA you will always build your command lines like this:

```
bwa <command> [options]
```

This means you first make the call to the binary ('bwa'), then you specify which command (method) you wish to use (e.g. 'index') then any options (i. e. arguments such as input files or parameters) used by the program to perform that command.

SAMtools(Li et al., 2009)

7 Read the overview of the SAMtools software on the project homepage at http://samtools.sourceforge.net/.

8 Download the latest version of the software package from http://sourceforge.net/projects/samtools/files/.

9 Unpack the tar file using:

```
tar xjf samtools-0.1.19.tar.bz2
```

This will produce a directory called "samtools-0.1.19" containing the files necessary to compile the SAMtools binary.

10 Move to this directory and compile using:

```
cd samtools-0.1.19
make
```

The compiled binary is called "samtools". You should find it within the same folder (samtools-0.1.19 in this example).

11 Add the SAMtools binary to your path to make it available on the command line. This completes the installation process.

12 Test your installation of SAMtools by opening a shell and running:

```
samtools
```

This should print out some version information as well as a list of commands. As the "Usage" line states, to use SAMtools you will always build your command lines like this:

```
samtools <command> [options]
```

This means you first make the call to the binary ('samtools'), then you specify which command (method) you wish to use (e.g. 'index') then any options (i. e.

arguments such as input files or parameters) used by the program to perform that command. This is the same convention as used by BWA.

HTSlib

13 Read the overview of the HTSlib software on the project homepage at https://github.com/samtools/htslib.

14 Download the latest version of the software package from https://github.com/samtools/htslib/archive/master.zip.

15 Unpack the tar file using:

```
tar xjf htslib-master.zip
```

This will produce a directory called “htslib-master” containing the files necessary to compile the HTSlib binary.

16 Move to this directory and compile using:

```
cd htslib-master
make
```

The compiled binary is called “htscmd”. You should find it within the same folder (htslib-master in this example).

17 Add the HTSlib binary to your path to make it available on the command line. This completes the installation process.

18 Test your installation of HTSlib by opening a shell and running:

```
htscmd
```

This should print out some version information as well as a list of commands. As the “Usage” line states, to use HTSlib you will always build your command lines like this:

```
htscmd <command> [options]
```

This means you first make the call to the binary (’htscmd’), then you specify which command (method) you wish to use (e.g. ’index’) then any options (i. e. arguments such as input files or parameters) used by the program to perform that command. This is the same convention as used by BWA and SAMtools.

Picard

19 Read the overview of the Picard software on the project homepage at http://picard.sourceforge.net/.

20 Download the latest version of the software package from http://sourceforge.net/projects/picard/files/.

21 Unpack the zip file using:

```
tar xjf picard-tools-1.88.zip
```

This will produce a directory called “picard-tools-1.88” containing the Picard jar files. Picard tools are distributed as pre-compiled Java executables (jar files) so there is no need to compile them.

22 Add the Picard directory to your path to make the tools available on the command line. This completes the installation process.

23 Test your installation of Picard by opening a shell and running:

```
java -jar AddOrReplaceReadGroups.jar -h
```

This should print out some version and usage information about the ’AddOrReplaceReadGroups.jar’ tool. At this point you will have noticed an important difference between BWA and Picard tools:

- To use BWA, we called on the BWA program and specified which of its internal tools we wanted to apply.
- To use Picard, we called on Java itself as the main program, then specified which jar file to use, knowing that one jar file = one tool. This applies to all Picard tools; to use them you will always build your command lines like this:

  ```
  java -jar <ToolName.jar> [options]
  ```

  Next we will see that GATK tools are called in yet another way. The reasons for how tools in a given software package are organized and invoked are largely due to the preferences of the software developers. They generally do not reflect strict technical requirements, although they can have an effect on speed and efficiency.

Genome Analysis Toolkit (GATK)(DePristo et al., 2011; McKenna et al., 2010)

24 Read the overview of the GATK on the project homepage at http://www.broadinstitute.org/gatk/about/.

25 Download the latest version of the software package from http://www.broadinstitute.org/gatk/download.

In order to access the downloads, you need to register for a free account on the GATK support forum at http://gatkforums.broadinstitute.org/. *You will also need to read and accept the license agreement before downloading the GATK software package. Note that if you intend to use the GATK for commercial purposes, you will need to purchase a license. See* http://www.appistry.com/gatk *for more information on the commercial licensing conditions.*

26 Unpack the tar file using:

```
tar xjf GenomeAnalysisTK-2.4-9.tar.bz2
```

This will produce a directory called "GenomeAnalysisTK-2.4-9-g532efad" containing the GATK jar file, which is called 'GenomeAnalysisTK.jar', as well as a directory of example files called 'resources'. GATK tools are distributed as a single pre-compiled Java executable so there is no need to compile them.

27 Add the GATK directory to your path to make the tools available on the command line. This completes the installation process.

28 Test your installation of GATK by opening a shell and running:

```
java -jar GenomeAnalysisTK.jar -h
```

This should print out some version and usage information, as well as a list of the tools included in the GATK. As the "Usage" line states, to use GATK you will always build your command lines like this:

```
java -jar GenomeAnalysisTK.jar -T <ToolName> [arguments]
```

This means you first make the call to Java itself as the main program, then specify the GenomeAnalysisTK.jar file, then specify which tool you want, and finally you pass whatever other arguments (input files, parameters etc.) are needed for the analysis.

RStudio IDE and the R libraries ggplot2(Wickham, 2009) and gsalib(DePristo et al., 2011)

29 Download the latest version of RStudio IDE from http://www.rstudio.com/ide/download/desktop. The webpage should automatically detect what platform you are running on and recommend the version most suitable for your system.

30 Follow the installation instructions. Binaries are provided for all major platforms; typically they just need to be placed in your Applications (or Programs) directory.

31 Open RStudio and type the following command in the console window:

```
install.packages(“ggplot2”)
```

This will download and install the ggplot2 library as well as any other library packages that ggplot2 depends on for its operation.

32 Now do the same thing to install the gsalib library:

```
install.packages(“gsalib”)
```

33 Finally, test your installation by executing the example R script provided in this unit’s resource bundle. You will need to:

- Navigate to the resource bundle: use the “Files” tab in the window located in the lower right quadrant of the RStudio window.
- Set the resource bundle as the working directory: click on “More” in the menu of the “Files” window and click on “Set As Working Directory” in the drop down menu that appears.
- Open the example file “cbpi_testplot.Rscript”: click on the file name in the “Files” window. The script text will be displayed in the upper left quadrant of the RStudio window.
- Run the script: click on “Run” in the menu bar of the open script file. If your installation works correctly, a new plot will be created and appear in the “Plots” window in the lower right quadrant of the RStudio window.

SUPPORT PROTOCOL 2

FROM BAM BACK TO FASTQ: REPROCESSING OLD DATA

Sometimes you need to work with sequence data that was assembled and/or processed by someone else. In such cases, you usually have no idea exactly how it was processed (which algorithm? with what parameters?), or it was done with older tools which you know were not as good as the ones available today. In this protocol, we show you how to revert a BAM file into raw FASTQ, which you can then process cleanly from the top of Basic Protocol 1.

Necessary Resources

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

    - Burrows-Wheeler Alignment Tool (BWA)
    - Picard Tools

- Hardware

  Same as described for Basic Protocol 1.

- Files

  You can find the following example files in the resource bundle associated with this unit.

    - The sequence data BAM file to revert (aln_reads.bam)

Reverting a BAM file

34 Shuffle the reads in the bam file so they are not in a biased order before alignment by running the following HTSlib command:

```
htscmd bamshuf -uOn 128 aln_reads.bam tmp >
shuffled_reads.bam
```

This creates a new BAM file containing the original reads, which still retain their mapping information, but now they are no longer sorted.

The aligner uses blocks of paired reads to estimate the insert size. If you don't shuffle your original bam, the blocks of insert size will not be randomly distributed across the genome, rather they will all come from the same region, biasing the insert size calculation. This is a very important step which is unfortunately often overlooked.

35 Revert the BAM file to FastQ format by running the following HTSlib command:

```
htscmd bam2fq -a shuffled_reads.bam > interleaved_reads.fq
```

This creates an interleaved FastQ file called "interleaved_reads.fq" containing the now-unmapped paired reads.

"Interleaved" simply means that for each pair of reads in your paired-end data set, both the forward and the reverse reads are in the same file, as opposed to having them in separate files.

36 Compress the FastQ file to reduce its size using the gzip utility:

```
gzip interleaved_reads.fq
```

This creates a gzipped FastQ file called "interleaved_reads.fq.gz". This file is ready to be used as input for Basic Protocol 1.

BWA handles gzipped fastq files natively, so you don't need to unzip the file to use it later on.

Note that if you're feeling adventurous, you can do all of the above with this beautiful one-liner, which will save you a heap of time that the program would otherwise spend performing I/O (loading in and writing out data to/from disk):

```
htscmd bamshuf -uOn 128 aln_reads.bam tmp | htscmd bam2fq -a - |
gzip > interleaved_reads.fq.gz
```

SUPPORT PROTOCOL 3

FIXING IMPROPERLY FORMATTED BAM FILES

You may occasionally receive sequence data from colleagues or from online data repositories in the form of BAM files that do not suit the requirements of the GATK tools. These requirements are that all BAM files should be properly indexed, sorted with reads in coordinate order, with duplicates marked, and all reads tagged with read groups. In Basic Protocol 1, we gave you streamlined instructions that show you how to perform the necessary processing in the least possible number of steps by combining some of the component operations. In this protocol, we will show you how to perform each of these steps independently of each other so that you need only apply the necessary formatting fix without having to redo everything. The end product of this protocol is a well-formatted BAM file that is ready to be pre-processed using GATK tools as described in Basic Protocol 1, steps X to Y.

Necessary Resources

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  - Picard Tools

- Hardware

  Same as described for Basic Protocol 1.

- Files

  You can find the following example files in the resource bundle associated with this unit.

  - The data to process: unsorted sequence reads in BAM format (unsorted_reads.bam)

    Note that all files must be able to pass strict validation of their respective format specifications.

37 Sort the aligned reads by running the following Picard command:

```
java -jar SortSam.jar \
INPUT=unsorted_reads.bam \
OUTPUT=sorted_reads.bam \
SORT_ORDER=coordinate
```

This creates a file called “sorted_reads.bam” containing the aligned reads sorted by coordinate.

38 Mark duplicate reads by running the following Picard command:

```
java -jar MarkDuplicates.jar \
INPUT=sorted_reads.bam \
OUTPUT=dedup_reads.bam
```

This creates a file called “dedup_reads.bam” with the same content as the input file, except that any duplicate reads are marked as such.

During the sequencing process, the same DNA molecules can be sequenced several times. The resulting duplicate reads are not informative and should not be counted as additional evidence for or against a putative variant. The duplicate marking process (sometimes called “dedupping” in bioinformatics slang) identifies these reads as such so that the GATK tools know to ignore them.

39 Add read group information by running the following Picard command:

```
java -jar AddOrReplaceReadGroups.jar \
INPUT=dedup_reads.bam \
OUTPUT=addrg_reads.bam \
RGID=group1 RGLB= lib1 RGPL=illumina RGPU=unit1
RGSM=sample1
```

This creates a file called “addrg_reads.bam” with the same content as the input file, except that the reads will now have read group information attached.

40 Index the BAM file by running the following Picard command:

```
java -jar BuildBamIndex \
INPUT=addrg_reads.bam
```

This creates an index file called “addrg_reads.bai”, which is ready to be used in Basic Protocol 1, from step X.

Since Picard tools do not systematically create an index file when they output a new BAM file (unlike GATK tools, which will always output indexed files), it is best to keep the indexing step for last.

SUPPORT PROTOCOL 4

ADDING VARIANT ANNOTATIONS WITH VariantAnnotator

Variant annotations are useful for all kinds of purposes such as filtering or performing various types of analyses on a call set. The GATK callers add certain standard annotations by default by the GATK callers, as well as any that we specified in the command line used to call variants. But what if you forget to specify an annotation, or you realize only later that a certain annotation would be useful? In this protocol, we show you how to add new annotations to variants without having to redo the calling step, using the VariantAnnotator tool. The end product of this protocol will be a VCF with additional annotations.

Necessary Resources

- Hardware

  Same as described for Basic Protocol 1.

- Software

  See Support Protocol 1 for detailed instructions on how to obtain and install this software.

  – Genome Analysis Toolkit (GATK)

- Files

  All files must be able to pass strict validation of their respective format specifications.

  – The sequence data in BAM format, processed as described in Basic Protocol 1 (reduced_reads.bam)

  – The call set to annotate in VCF format, produced as described in Basic Protocol 2 (raw_variants.vcf)

  – The human reference genome in FASTA format (reference.fa)

  Adding variant annotations with VariantAnnotator Note that for the re-annotation process, the program requires that you provide the original sequence dataset from which the call set was derived.

1 Choose the annotations to add.

Some annotations can only be calculated under certain conditions, which can be general, such as a minimum number of samples (e.g. the InbreedingCoefficient requires at least 10 samples) or local, such as a certain state of zygosity (e.g. the MappingQualityRankSumTest can only be calculated at homozygous sites). Keep in mind also that many of the built-in annotations available in the GATK make the assumption that samples are diploid, because the program was originally developed to analyze human genomes exclusively. Please always check the relevant documentation before attempting to use a new annotation.

- MappingQualityZero (MQ0)

  This is the total count (across all samples) of reads that have a mapping quality of zero. This can be a useful annotation for diagnostic purposes.

- SpanningDeletions

  This is the fraction of reads containing spanning deletions at this site. This annotation can also be useful for diagnostic purposes.

2 Call variants in your sequence data by running the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
-T VariantAnnotator \
-R reference.fa \
-I reduced_reads.bam \
-V raw_variants.vcf \
-L raw_variants.vcf \
-A MQ0 \
-A SpanningDeletions \
-o raw_reannotated_variants.vcf
```

This creates a VCF file called “raw_reannotated_variants.vcf”, containing all the original sites from the “raw_variants.vcf” file, but now the variants are annotated with the new annotations where applicable.

LITERATURE CITED

DePristo MA, Banks E, Poplin R, Garimella KV, Maguire JR, Hartl C, Philippakis AA, del Angel G, Rivas MA, Hanna M, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature Genetics. 2011; 43:491–498. [PubMed: 21478889]

Durbin RM, Altshuler DL, Durbin RM, Abecasis GR, Bentley DR, Chakravarti A, Clark AG, Collins FS, La Vega, De FM, Donnelly P, et al. A map of human genome variation from population-scale sequencing. Nature. 2010; 467:1061–1073. [PubMed: 20981092]

Fisher RA. JSTOR: Journal of the Royal Statistical Society, Vol. 85, No. 1 (Jan., 1922), pp. 87–94. Journal of the Royal Statistical Society. 1922

Altshuler DM, Gibbs RA, Peltonen L, Altshuler DM, Gibbs RA, Peltonen L, Dermitzakis E, Schaffner SF, Yu F, et al. International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. Nature. 2010; 467:52–58. [PubMed: 20811451]

Li H, Durbin R. Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics (Oxford, England). 2010; 26:589–595.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. Bioinformatics (Oxford, England). 2009; 25:2078–2079.

Mann HB, Whitney DR. On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics. 1947

McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research. 2010; 20:1297–1303. [PubMed: 20644199]

Mills RE, Luttig CT, Larkins CE, Beauchamp A. An initial map of insertion and deletion (INDEL) variation in the human genome. Genome …. 2006

Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K. dbSNP: the NCBI database of genetic variation. Nucleic acids research. 2001; 29:308–311. [PubMed: 11125122]

Wickham, H. ggplot2: Elegant Graphics for Data Analysis. Springer Publishing Company, Incorporated; 2009.

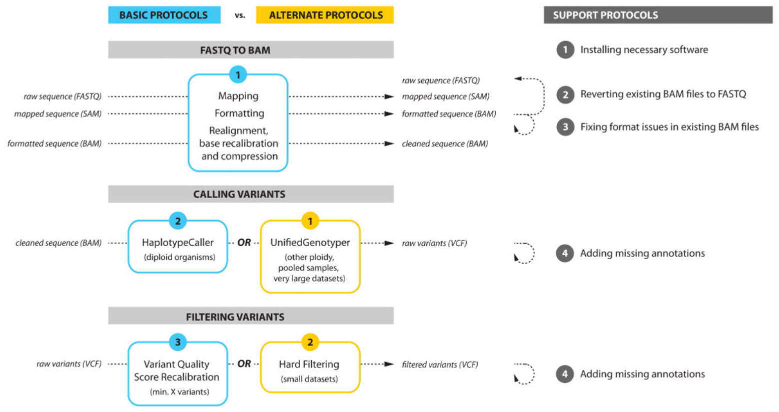
BASIC PROTOCOLS
vs.
ALTERNATE PROTOCOLS
SUPPORT PROTOCOLS
FASTQ TO BAM
1 Installing necessary software
raw sequence (FASTQ)
mapped sequence (SAM)
formatted sequence (BAM)
Mapping
Formatting
Realignment, base recalibration and compression
raw sequence (FASTQ)
mapped sequence (SAM)
formatted sequence (BAM)
cleaned sequence (BAM)
2 Reverting existing BAM files to FASTQ
3 Fixing format issues in existing BAM files
CALLING VARIANTS
cleaned sequence (BAM)
HaplotypeCaller (diploid organisms)
OR
UnifiedGenotyper (other ploidy, pooled samples, very large datasets)
raw variants (VCF)
4 Adding missing annotations
FILTERING VARIANTS
raw variants (VCF)
Variant Quality Score Recalibration (min. X variants)
OR
Hard Filtering (small datasets)
filtered variants (VCF)
4 Adding missing annotations

Figure 1.
Strategic planning workflow for the protocols included in this unit.