# Chapter 21

# Bioinformatics Analysis of Whole Exome Sequencing Data

**Peter J. Ulintz, Weisheng Wu, and Chris M. Gates**

## Abstract

This chapter contains a step-by-step protocol for identifying somatic SNPs and small Indels from next-generation sequencing data of tumor samples and matching normal samples. The workflow presented here is largely based on the Broad Institute's "Best Practices" guidelines and makes use of their Genome Analysis Toolkit (GATK) platform. Variants are annotated with population allele frequencies and curated resources such as GnomAD and ClinVar and curated effect predictions from dbNSFP using VCFtools, SnpEff, and SnpSift.

**Key words** Next-generation sequencing, Cancer research, Exome sequencing, Genome sequencing, Clinical genomics, Somatic variant detection, Variant annotation

## 1 Introduction

Next-generation sequencing (NGS) experiments continue to facilitate a dramatic shift in the way genetic variation is detected and utilized. While it is now feasible to sequence the entire genome of patients or research samples, the cost of doing so at a depth necessary to comprehensively profile mutations is still somewhat prohibitive. Instead, researchers or clinicians often seek to limit sequencing to a smaller genomic target, permitting a more comprehensive analysis of the selected regions. Whole exome sequencing (WES) implements a capture-based enrichment of the protein coding regions of genomic DNA, utilizing a set of oligonucleotide hybridization probes that target known exon sequences. Utilizing this method has permitted the profiling of many cancer samples and tumors and has led to the characterization of new therapeutic targets, a canonical example being EGFR variants in colorectal cancer patients without confounding KRAS mutations treated using Cetuximab [1].

The most common application of WES is mutational analysis: the detection of single-nucleotide variants (SNVs) or small insertions and deletions (Indels). Other applications are the profiling of

Sami N. Malek (ed.), *Chronic Lymphocytic Leukemia: Methods and Protocols*, Methods in Molecular Biology, vol. 1881,
https://doi.org/10.1007/978-1-4939-8876-1_21, 

copy-number variations (CNVs) and the detection of structural variations (SVs, often defined as genomic rearrangements larger than 50 bp). The focus of this chapter is on somatic variant detection. Given the data generated by WES of a matched cancer/normal sample pair, we present a complete analytic workflow for detecting and characterizing the somatic mutations present in the cancer sample.

Overall, we'll be following a protocol that has been established by a leading institution in this domain, the Broad Institute, largely following what is known as their "Best Practices" guidelines for variant detection [2]. We'll be using tools available in the current release of Broad's Genome Analysis Toolkit (GATK) [3]. This toolkit provides many of the tools necessary for a genomic analysis in one downloadable package. GATK has recently been open-sourced, is relatively well documented, and represents a somewhat standardized protocol in this field based on its popularity in the literature.

A significant number of algorithms and software tools have been developed for detecting variants in NGS sequencing data [4–8] (*see* **Note 1**). The detection of somatic variants is typically performed using algorithms and software tools specialized for the task, since cancer samples often exhibit properties that may confound a standard germline variant caller such as the presence of normal or non-aberrant cells, copy number events, or tumor heterogeneity and subclonality [9, 10]. Moreover, the availability of a matching normal sample provides the opportunity to accommodate an entirely new type of information: rather than simply assessing differences between a test sample and the reference genome, variation can be assessed with respect to both the reference and the matching normal. Historically, although approaches have been articulated based on analyzing tumor and matching normal samples independently following by a subtraction-based strategy [11, 12], more sophisticated algorithms involving simultaneous analysis of both matched samples using joint probabilistic models have been more widely utilized [13–15]. Such models, in addition to detecting the presence of a variant at a specific locus (with an associated likelihood or set of scores), can also classify a variant in a cancer sample as either germline or somatic with a second measure of likelihood. Somatic-specific algorithms can be permitted to relax certain constraints that exist in germline detection approaches such as ploidy assumptions, or the assumption that a heterozygous variant will be present at a minimum allele fraction, which contribute to increased sensitivity.

The workflow presented here is based the latest version (at the time of this writing) of the Mutect2 somatic variant caller, largely following the Broad GATK4 Somatic SNVs + Indels Best Practices workflow. The particular version of the toolset presented here is based on the January 9, 2018 GATK4.0.0.0 release, the first major

GATK release since March 2014. As such, the main workflow we present here includes tool versions which are newly released or listed as research-only. For example, although the first release of the current incarnation of the Mutect2 algorithm was in November of 2015, it is still listed as beta. The bioinformatics tools used in genomic analysis evolve rapidly. It is likely that at the time of publication and reading of this chapter, some of the commands in this workflow will have changed. The best resources for finding current versions of many of these commands is the Broad GATK website itself [16], or the SnpEff/SnpSift SourceForge repository documentation [17].

Users of this chapter may prefer production-grade tools, or may be interested in using a different class of variant callers. We are therefor also describing a supplementary workflow based on a second popular caller: VarScan Somatic [18]. This workflow is demonstrative of a class of variant callers that do not perform local haplotype assembly, thus benefitting from a specific GATK-based step known as Indel Realignment which would be superfluous in the main workflow. The supplemental workflow is currently valid and quite common, but will likely be supplanted by a version of the main workflow during the useful life of this chapter.

## 2 Materials

### 2.1 Files

A listing of the files used in this chapter will be provided below, and they are introduced here. Instructions for downloading the various auxiliary files are provided in Subheading 3.1.3.

The starting point for the workflow outlined in this chapter are files of raw human sequencing data in FASTQ format [19]. FASTQ is an ASCII text-based format that stores the nucleotide sequence of individual reads as well as quality information for each nucleotide in a four line per read format. Useful characteristics of the sequencing data to know prior to an analysis are: (a) the instrument on which the data were generated; (b) the length of each read; and (c) whether the reads were generated in a single-end or paired-end experiment. For whole exome sequencing (WES) experiments, paired-end data are preferred over single-end experiments if permitted by the sequencing platform. For this chapter, we will assume data are paired-end 150 bp reads generated on an Illumina instrument. Modifications to the protocol may be necessary for other data types (*see* **Note 2**). We will assume that a matched tumor and normal pair of files are available for this workflow, and provide an example of detecting somatic mutations using the paired sample set. The Mutect2 algorithm used here does have the ability to call variants in an unmatched cancer or tumor sample, and may provide some advantage over a standard germline variant detection algorithm such as HaplotypeCaller [20] or FreeBayes [21] when doing

so, but the tumor-only workflow is currently an unsupported use case for this tool. As such, although we'll comment on tumor-only usage, we do not provide a full workflow for this mode of analysis.

Although not strictly required, an important additional item of data to have available is a file (or files) listing the genomic locations of the targeted regions of the exome-capture assay. WES is a DNA or RNA hybridization-based assay. The various exome capture platforms consist of sets of probes, each targeting a specific genomic location. In the case of WES, the probes target the exonic regions of known genes (*see* **Note 3**). Provided with each platform is typically a file indicating either the specific locations of each probe and/or the genomic footprint of all overlapping probes. These files are usually provided in a simple BED format (*see* **Note 4**). The file can be utilized by several tools in the workflow for calculating coverage and other quality control measures, or to limit variant calls to targeted regions.

Several of the steps in this protocol make use of publicly available datasets to perform various tasks. Most of these files are distributed as part of the GATK Resource Bundle, but may also be available directly from native sources.

1. Raw sequencing reads in FASTQ format, gzipped: **Normal_01_R1.fastq.gz**, **Normal_01_R2.fastq.gz, Tumor_01_R1.fastq.gz, Tumor_01_R2.fastq.gz.**
2. (Optional but desirable) A panel of normal samples (PoN) VCF file (*see* Subheading 3.4.1 on constructing a PoN): **Project_PoN.vcf.gz**, along with its index (*see* **Note 5**).
3. Adapters file for trimming software: **TruSeq3-PE-2.fa.** This FASTA file should be available with the Trimmomatic software package, or can be constructed to match specific adapter versions (*see* **Note 6**).
4. Files from the GATK Resource Bundles (*see* Subheading 3.2):
   (a) Reference genome fasta file: **ucsc.hg19.fasta.gz.**
   (b) Index file for read aligner (BWA): **ucsc.hg19.fasta.fai.gz.**
   (c) Sequence dictionary: **ucsc.hg19.dict.gz.**
   (d) dbSNP variant file: **dbsnp_138.hg19.excluding_sites_after_129.vcf.gz.**
   (e) File of known human INDEL locations: Mills INDELs**: Mills_and_1000G_gold_standard.indels.hg19.vcf.gz.**
   (f) File of known human INDEL locations: **1000G_phase1.indels.hg19.vcf.gz.**
   (g) Population variant file, GnomAD: **af-only-gnomad.raw.sites.b37.vcf.gz** (*see* **Note 7**).
5. Exome bait and/or capture target BED files and interval files (*see* **Note 8**). The BED files are typically obtained from the

sequencing center or from the vendor of the exome capture kit that is being used, and will be named following the vendor's conventions. We'll assume the following names: **exome_capture.target.bed, exome_capture.bait.bed**.

6. (Optional but desirable) The dbNSFP annotation database (*see* Subheading "Install Software for Variant Calling").

### 2.2 Software

The workflow for analysis of WES data presented in this chapter consists of a set of processing steps, each enabled by one or several freely-available software tools. Several steps are processor and/or memory-intensive, and several of the tools permit multithreaded operations that may benefit from the availability of multiple cores; such tools and steps will be noted. For the workflow described below, we recommend a system with at least four cores and 16GB RAM.

*See* Table 1 for a listing of the packages and tools used in this chapter. We provide version numbers for each tool as a reference to a set of compatible versions at the time of writing. As a general rule, using current versions of software tools is recommended. However, the specifics of commands and parameters can change from version to version for any given tools, and we cannot guarantee compatibility of future versions of all tools.

## 3 Methods

### 3.1 Setup

#### 3.1.1 Folder Setup

We will outline the workflow method as a series of commands in a terminal or console session on a standard unix workstation or server. We will assume a main working folder which is designated as $WES, and all files generated or used will be placed in this folder or in specified subfolders of this main folder. Raw data from the sequencing instrument will be placed in a subfolder named `input`; the reference genome and other public data resources will be placed in `reference`; and specific auxiliary scripts and tools in `scripts`. A `tmp` folder will be created as well (*see* **Note 9**).

1. Set up an environment variable for the main folder. This may be done using the "export" command on most linux platforms:

```
export WES=/path/to/project/root
```

2. Create the project home folder structure:

```
cd $WES
mkdir -p input reference scripts tmp
```

**Table 1**
**Tools for somatic variant identification and annotation using Mutect2**

| Package (version) | Tool | Function as used in this document |
|---|---|---|
| BWA (0.7.15) | `index` | Index database sequences in the FASTA format |
| | `mem` | Align reads to reference genome |
| Conda (4.3.21) | `config` | Install and manage packages and environment |
| | `create` | Create a new conda environment |
| FastQC (0.11.5) | `fastqc` | Assess sequencing read quality |
| GATK (4.0.0.0) | `AnalyzeCovariates` | Evaluate and compare base quality score recalibration (BQSR) tables |
| | `ApplyBQSR` | Apply base quality score recalibration |
| | `BaseRecalibrator` | Detect systematic errors in base quality scores |
| | `BedToIntervalList` | Convert a BED file to a Picard Interval List |
| | `BQSR.R` | Generate plots for visualizing the quality of a BQSR run |
| | `CalculateContamination` | Calculate the fraction of reads coming from cross-sample contamination |
| | `CollectHsMetrics` | Collect hybrid-selection (HS) metrics for an alignment file |
| | `CollectSequencingArtifactMetrics` | Collect metrics to quantify single-base sequencing artifacts |
| | `CreateSequenceDictionary` | Create a sequence dictionary for a reference sequence |
| | `CreateSomaticPanelOfNormals` | Make a panel of normals for use with Mutect2 |
| | `FilterByOrientationBias` | Filter Mutect2 somatic variant calls using orientation bias |
| | `FilterMutectCalls` | Filter somatic SNVs and indels called by Mutect2 |
| | `GetPileupSummaries` | Calculate pileup statistics for inferring contamination |
| | `MarkDuplicates` | Locate and tag duplicate reads in an alignment file |
| | `Mutect2` | Call somatic short variants via local assembly of haplotypes |
| | `SortSam` | Sort, compress and index alignment |
| Jacquard (0.42) | `expand` | Expand a VCF file into a TXT |

(continued)

**Table 1**
**(continued)**

| Package (version) | Tool | Function as used in this document |
|---|---|---|
| SAMtools (1.2) | `faidx` | Index reference sequence in the FASTA format |
| | `flagstat` | Report alignment statistics |
| SnpEff (4.3T) | `database` | List all supported databases |
| | `snpeff` | Annotate genomic variants and predict functional effect |
| SnpSift (4.3T) | `dbnsfp` | Annotate genomic variants using dbNSFP |
| | `extractFields` | Extract fields from a VCF file to a TXT |
| Trimmomatic (0.36) | `trimmomatic-0.36.jar` | Trim reads for quality |
| VCFtools (0.1.15) | `fill-fs` | Annotate VCF with flanking sequence |

### 3.1.2 Software Setup

Install Software for Variant Calling

3. Download and install the Conda package manager. Installation instructions for Conda may be found at this URL: https://conda.io/docs/user-guide/install/index.html. Miniconda should be fine, and Python2 is preferred for compatibility with the annotation software.
4. Configure Conda channels.

```
conda config --add channels r
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

5. Create a new Conda environment:

```
conda create --name wes \
   bwa fastqc gatk4 R samtools trimmomatic
```

6. Activate the conda environment:

```
source activate wes
```

7. Download the Broad GATK Base Quality Score Recalibration plotting script:

```
cd $WES/scripts
wget -np -nd \
  https://github.com/broadgsa/gatk/blob/master/public/gatk-
engine/src/main/resources/org/broadinstitute/gatk/engine/re-
calibration/BQSR.R
```

Install Software for Annotation

8. Extend the existing Conda environment with new annotation packages.

```
conda install -c bioconda python=2 \
   SnpEff SnpSift \
   tabix \
   perl-vcftools-vcf
```

9. Install Jacquard, a tool to facilitate VCF parsing and manipulation.

```
pip install jacquard
```

10. Install the VCFtools:fill-fs utility script: the Conda install above will establish VCFtool's core Perl modules, but does not install the many useful Perl utilities. The lines below fetch and setup a minimal install of the fill-fs Perl program (*see* **Note 10**).

```
cd scripts
wget --output-document vcftools.zip \
   https://github.com/vcftools/vcftools/zipball/master
unzip vcftools.zip && rm vcftools.zip
find vcftools* -name 'fill-fs' | xargs -I {} ln -s {}
cd $WES
```

11. Install SnpEff helper scripts (*see* **Note 11**).

```
cd scripts
rp="import __future__,sys,os.path;print(os.path.realpath(sys.
argv[1]))" &&
   alias realpath='python -c "$rp"'
export SNPEFF_SCRIPTS=$(dirname $(realpath $(which snpeff)))/
scripts
find $SNPEFF_SCRIPTS \
   | egrep 'vcfAnnFirst|vcfEffOnePerLine' \
   | xargs -I {} ln -s {}
cd $WES
```

12. Install SnpSift helper script (optional). An optional annotation step depends on the dbNSFP database; this helper script is required to build that database.

```
cd scripts
snpeff_source=https://raw.githubusercontent.com/pcingola/
SnpEff/master
wget ${snpeff_source}/scripts_build/dbNSFP_sort.pl
chmod ug+x dbNSFP_sort.pl
cd $WES
```

### 3.1.3 Genomic Reference Setup

13. Download the GATK Resource Bundle files:

```
cd $WES/reference
wget -r -np -nd ftp://gsapubftp- anonymous@ftp.broadinstitute.
org/bundle/hg19
wget -np -nd ftp://gsapubftp-anonymous@ftp.broadinstitute.
org/bundle/beta/Mutect2/af-only-gnomad.raw.sites.b37.vcf.gz
wget -np -nd ftp://gsapubftp-anonymous@ftp.broadinstitute.
org/bundle/beta/Mutect2/af-only-gnomad.raw.sites.b37.vcf.gz.
tbi
```

14. Download Trimmomatic adapter files:

```
wget -np -nd \
https://github.com/timflutre/trimmomatic/tree/master/adap-
ters/TruSeq3-PE-2.fa
```

15. Unzip the reference genome FASTA file and its index.

```
gzip -d ucsc.hg19.fasta.gz
gzip -d ucsc.hg19.fasta.fai.gz
```

16. Index the reference genome:

```
bwa index -a bwtsw -p ucsc.hg19 reference/ucsc.hg19.fasta
```

17. Generate FASTA index. The result of these commands will be a set of files with the root name "ucsc.hg19.fasta":

```
samtools faidx reference/ucsc.hg19.fasta
```

18. Generate target interval files from BED files for use with Picard-based tools (*see* **Note 12**):

```
cd $WES/input
gatk-launch BedToIntervalList \
   -I input/exome_capture.target.bed \
   -O exome_capture.target.intervals \
   --SEQUENCE_DICTIONARY reference/ucsc.hg19.dict
```

19. (optional but desirable) Install the dbNSFP annotation database. The dbNSFP database is a clearinghouse of annotation data aggregated from independent source databases (*see* **Notes 13** and **14**). Note that dbNSFP is a large database: it requires several hours to download and several steps to prepare it for use.

    This download may take a few hours; if this command is interrupted, you can re-execute it and it will resume downloading where it left off.

```
cd reference && mkdir -p dbNSFP/tmp && cd dbNSFP/tmp
wget --continue ftp://dbnsfp:dbnsfp@dbnsfp.softgenetics.com/
dbNSFPv3.5a.zip
```

    Expand the downloaded zip. (The ~20 Gb zip file will expand to ~130Gb of source files in about 20 min).

```
unzip dbNSFPv3.5a.zip
```

    The latest version of dbNSFP is built for GRCh38/hg38 genome builds; to use the database with GRCh37/hg19 data one needs to remap the variant coordinates (using a script provided by SnpEff/SnpSift maintainers) and then re-sort and re-index the database. This transformation takes about 2 h (*see* **Note 15**).

```
version="3.5a"
cat dbNSFP${version}_variant.chr* \
    | ../../scripts/dbNSFP_sort.pl 7 8 \
    > dbNSFP${version}_hg19.txt
bgzip dbNSFP${version}_hg19.txt \
   && tabix -s 1 -b 2 -e 2 dbNSFP${version}_hg19.txt.gz
```

### 3.2 Preprocessing A: Basic Preprocessing and QC

An outline of the overall workflow is shown in Fig. 1.

1. Prior to any actual processing, perform quality diagnostics on the sequencing data. The following command will run Fastqc on the raw FASTQ files [22, 23]:

```
fastqc -o . \
  --extract \
  --format fastq \
  input/Tumor_01_R1.fastq.gz
```

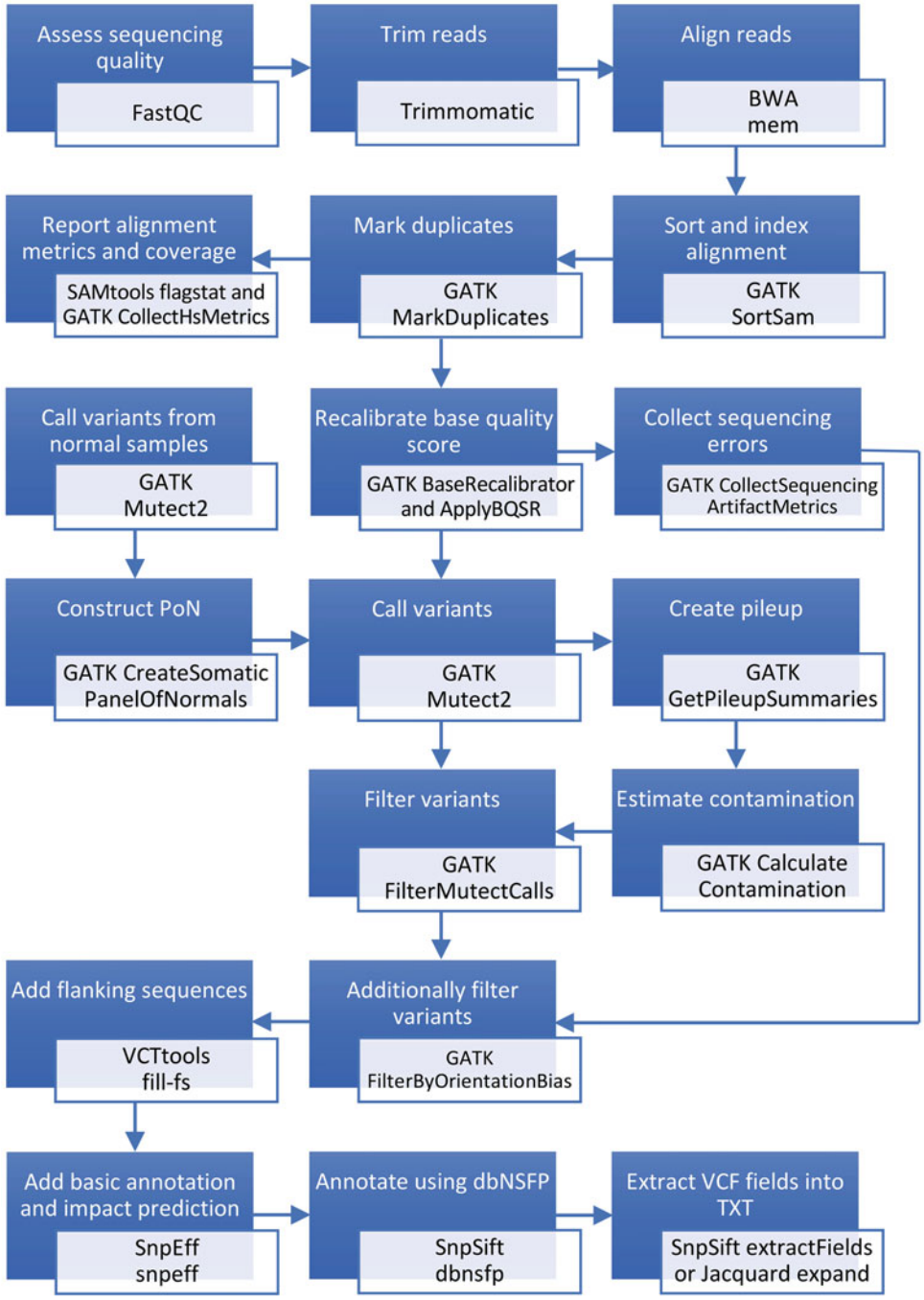


**Fig. 1** Workflow of somatic variant identification and annotation using GATK4 and Mutect2

(repeat for three remaining fastq.gz files.)Each command will produce a compressed (zip) file and a subdirectory of results, as well as an html file. Check the resulting FastQC reports. In particular,

check whether the quality of bases at the 5′ or 3′ ends of reads falls below tolerable thresholds. If so, modify the following step to trim corresponding bases from the raw read data (*see* **Note 16**).

2. Trim the FASTQ files. The following command uses the Trimmomatic tool [24], which will take as input paired gzipped FASTQ files and produce four output files: two paired files of trimmed output and two files containing reads for which only a single read of a pair survived. The ILLUMIACLIP portion of the command specifies several items delimited by spaces and colons, including the location of a FASTQ file containing all adapter sequences, low-quality base trimming parameters, and a minimum read length parameter (*see* **Note 17**).

```
trimmomatic PE \
   -phred33 \
   input/Tumor_01_R1.fastq.gz \input/Tumor_01_R2.fastq.gz \Tu-
mor_01_R1_trimmed.fastq \Tumor_01_R1_unpaired.fastq \
   Tumor_01_R2_trimmed.fastq \Tumor_01_R2_unpaired.fastq \IL-
LUMINACLIP:reference/TruSeq3-PE-2.fa:2:30:10
   LEADING:10 TRAILING:10 SLIDINGWINDOW:5:0 MINLEN:25
```

If multiple cores are available, the "-threads" parameter may be used (e.g., `-threads 4`).

(repeat for the normal samples)

3. Align the reads to the reference genome using BWA mem [25]. This command takes paired, trimmed read files as input as well as the location of the reference genome and generates a single-alignment (SAM) file.

```
bwa mem \
   -R '@RG\tID:Tumor_01\tLB:Tumor_01\tSM:Tumor_01\tPL:ILLUMI-
NA' \
   -M reference/ucsc.hg19.fasta \
   Tumor_01_R1_trimmed.fastq  \
   Tumor_01_R2_trimmed.fastq \
   > Tumor_01.sam
```

If multiple cores are available, the -t parameter may be used. Note that we're populating the Sample "SM:" field in the Read Group (RG) section of the BAM header with a sample name. This name will be used in a later step as an argument to the Mutect2 variant caller.

(repeat for normal samples)

4. Sort, compress, and index the alignment file. This step will result in a single, sorted, indexed BAM file.

```
gatk-launch SortSam \-I Tumor_01.sam \
   -O Tumor_01.bam \
   --SORT_ORDER coordinate \
   --VALIDATION_STRINGENCY LENIENT \
   --CREATE_INDEX true \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

Note that once BAM files have been created, the `Tumor_01.sam` and `Normal_01.sam` files can be removed. They can be large and it may be desirable to save space. SAM files can be regenerated from their BAM file equivalents at any time.

5. Mark duplicate reads from the alignment file. For any set of paired reads that have precisely the same coordinates when aligned to the reference genome, which are often PCR duplicates, only the highest-scoring read is retained; the rest are marked as duplicates (*see* **Note 18**).

```
gatk-launch MarkDuplicates \
   -I Tumor_01.bam \
   -O Tumor_01_mkdp.bam \
   --REMOVE_DUPLICATES false \
   --METRICS_FILE Tumor_01_dup_metrics.txt \
   --CREATE_INDEX true \
   --VALIDATION_STRINGENCY LENIENT \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

6. Generate basic metrics on the de-duplicated BAM files (optional). The `samtools flagstat` tool will assemble overall read counts, pairing, mapping, and deduplication metrics on BAM files. The `flagstat` tool is described at the following location: http://www.htslib.org/doc/samtools.html.

```
samtools flagstat Tumor_01_mkdp.bam > Tumor_01_mkdp.bam.flagstat
```

(repeat for the normal sample)

7. Generate coverage data and capture metrics (optional). There are several options for profiling coverage across the target region of interest in an experiment, including `bedtools coverage` or the GATK `CollectFragmentCounts` tool, or even the GATK `DepthOfCoverage` tool if using an older version of the toolkit. Here, we will calculate a set of QC metrics using the Picard/GATK `CollectHsMetrics` tool. The tool takes as input the target and bait intervals files (*see* **Notes 8** and **19**). The output of the tool will be a tab-delimited text file with

various QC metrics, such as mean and median target coverages, the percentage of off-bait reads, and the percentage of the target that achieves particular coverage depths (e.g., 20×, 50×, 100×).

```
gatk-launch CollectHsMetrics  \
   -I Tumor_01_mkdp.bam \
   -O Tumor_01_mkdp.HSmetrics.tsv \
   --BAIT_INTERVALS exome_capture.bait.intervals \
   --TARGET_INTERVALS exome_capture.target.intervals \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

### 3.3 Preprocessing B: Base Quality Score Recalibration

These steps are performed using the Broad Institute's Genome Analysis Toolkit. Base Quality Score Recalibration (BQSR) uses a machine learning framework to model and correct systematic base scoring errors that the sequencing instrument algorithms may generate in particular regions of the genome such as homopolymer runs. It works in a two-pass manner, first building a model over all bases in the dataset as well as a set of known variants and writing the model to a table. The second pass actually applies the learned model to correct per-base alignment quality scores. BQSR is important in that variant calling algorithms make strong use of per-base quality scores, and is appropriate to use regardless of the downstream variant caller (*see* **Note 20**).

1. Generate recalibration model:

```
gatk-launch BaseRecalibrator \
   -I Tumor_01_mkdp.bam \
   -O Tumor_01_bqsr.table \
   -R reference/ucsc.hg19.fasta \
     --known-sites  reference/dbsnp_138.hg19.excluding_sites_-
after_129.vcf \
   --known-sites reference/1000G_phase1.indels.hg19.vcf \
    --known-sites  reference/Mills_and_1000G_gold_standard.in-
dels.hg19.vcf \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

2. Actually apply recalibration on reads overlapping the target intervals:

```
gatk-launch ApplyBQSR \
   -I Tumor_01_rmdp.bam \
   -O Tumor_01_recal.bam \
   -R reference/ucsc.hg19.fasta \
```

```
   --bqsr-recal-file Tumor_01_bqsr.table \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

At this point, we have recalibrated alignment (BAM) files ready for variant calling. It may be desirable to repeat **step 1** of this section on these recalibrated BAM files to obtain QC metrics as follows.

3. Build recalibration model on the recalibrated BAM file for comparison (optional):

```
gatk-launch BaseRecalibrator \
   -I Tumor_01_recal.bam \
   -O Tumor_01_postbqsr.table \
   -R reference/ucsc.hg19.fasta \
    --known-sites reference/dbsnp_138.hg19.excluding_sites_-
after_129.vcf \
   --known-sites reference/1000G_phase1.indels.hg19.vcf \
    --known-sites reference/Mills_and_1000G_gold_standard.in-
dels.hg19.vcf \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for the normal sample)

4. Use `AnalyzeCovariates` to compare the pre- and post-BQSR tables (optional) (*see* **Note 21**):

```
gatk-launch AnalyzeCovariates \
   --before-report-file Tumor_01_bqsr.table \
   --after-report-file Tumor_01_postbqsr.table \
   --intermediate-csv-file Tumor_01_BQSR.analyzecovariates.csv \
   -R reference/ucsc.hg19.fasta \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
Rscript scripts/BQSR.R \
   Tumor_01_BQSR.analyzecovariates.csv \
   Tumor_01_bqsr.table \
   Tumor_01_bqsr_plots.pdf
```

### 3.4 Variant Calling and Filtering

As mentioned above, one way of distinguishing variant calling algorithms is as germline algorithms which identify all mutations in a single-alignment file, or as somatic algorithms that typically take as input paired tumor and normal samples. Somatic algorithms, in addition to detecting variants in each of the two samples, will also classify detected mutations in the tumor sample as somatic vs germline using data from both samples at each variant locus. Here, we're showing a workflow using the GATK4 Mutect2 somatic algorithm (*see* **Note 22**). Mutect2's main mode is as a

Tumor/Normal variant caller, although it can call variants in tumor-only mode for samples without matching normal (although this mode is not currently recommended, as mentioned below).

The Mutect2 algorithm has undergone significant refinement since its original introduction [15]. It now inherits a local haplotype reassembly framework from HaplotypeCaller (HC) but relaxes the ploidy constraints present in HC to accommodate copy number change [20, 26–29]. It can accommodate data from a (usually public) germline variant resource for which population allele frequencies are available, as well as an unmatched Panel of Normals (PoN) dataset. It permits more fine-grained control over both germline and somatic modeling parameters. Mutect2 also accommodates population variant data files such as those generated by community projects such as dbSNP, 1kG, ExAC, and GnomeAD [29–32]; it can adjust the likelihood that a variant is somatic based on how frequently a specific variant is present in the population databases. See the Broad GATK online documentation for up-to-date information on this tool.

#### *3.4.1 Create a Panel of Normals*

Mutect2 can utilize a dataset of variants called from normal samples to assist in detecting somatic mutations. The dataset, called a Panel of Normals (PoN), is used by Mutect2 to profile systematic technical errors that may be present in datasets generated using the same experimental protocol. A variant detected across multiple samples may be an artifact or it may be a common germline variant; whichever it is, such variants are useful information for detecting somatic variants.

A PoN would not typically be constructed for an individual project; it would be a resource that is generated for use in multiple projects. It is best to assemble the PoN using as many normal samples as are available, but even a smaller PoN provides useful data to the caller. The normal samples used to compose the PoN should have been generated in as similar a manner as possible to the tumor samples being analyzed. They should have been generated on the same instrument using the same chemistry in the same sequencing lab, the same sequencing cycles (e.g., paired-150 bp), analyzed using the same workflow, and using the same genomic target. The normal samples from matched tumor/normal (T/N) pairs should not be used to generate a PoN that will be used to analyze those same pairs; the PoN needs to be created from an unrelated sample set generated using the same experimental protocol. If the only normal samples available are the matched ones, avoid using a PON and proceed with variant calling using the matched T/N pairs.

1. (If necessary) Generate analysis-ready BAM files for each normal sample to be used in the PoN. Follow workflow steps in Subheadings 3.2 and 3.3 to produce recalibrated BAM files.

2. Run Mutect2 on each normal sample as if it were a tumor sample to create a variant call set for the PoN:

```
gatk-launch Mutect2 \
   -I PoN_Normal_01_recal.bam \
   -O PON_Normal_01.Mutect2.forPON.vcf.gz \
    -R reference/ucsc.hg19.fasta \   --tumor-sample PoN_Nor-
mal_01 \--intervals reference/exome_capture.target.bed \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

(repeat for all normal PON samples)

3. Construct the PoN. Specify each normal VCF file generated in **step 2**.

```
gatk-launch CreateSomaticPanelOfNormals \
   --vcfs PON_Normal_01.Mutect2.forPON.vcf.gz \
   --vcfs PON_Normal_02.Mutect2.forPON.vcf.gz \
   --vcfs PON_Normal_03.Mutect2.forPON.vcf.gz \
   -O Project_PoN.vcf.gz \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

### 3.4.2 Call Variants

4. We can now call variants on the aligned reads. In this example, we will specify the command for a T/N pair. With significant caution, Mutect2 can also call variants on a tumor sample in the absence of a matching normal by removing references to normal in the command (*see* **Note 23** for an example tumor-only command, and **Note 24** for important caveats). If a PoN is absent, simply remove the respective argument. Mutect2 can also export the reassembled alignment file using the `--bam-output` argument if desired for QC and visualization. Note that the sample names specified to the --tumor-sample and --normal_sample arguments need to be the ones specified in the BAM header (*see* Subheading 3.2, **step 3**).

```
gatk-launch Mutect2 \
   -R reference/ucsc.hg19.fasta \
   -I Tumor_01_recal.bam \
   --tumor-sample Tumor_01 \
   -I Normal_01_recal.bam \
   --normal-sample Normal_01 \
   --intervals exome_capture.target.bed \
   -O Tumor_Normal_01.Mutect2.raw.vcf.gz \
  --germline-resource reference/af-only-gnomad.raw.sites.b37.
vcf \
   --panel-of-normals Project_PoN.vcf.gz \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

The resulting VCF file is a raw, unfiltered list of variants. There are data for both the tumor and the normal sample in the VCF file (*see* **Note 25**).

### 3.4.3 Filter Variants

Unlike earlier MuTect and MuTect2 versions, variants detected by the base Mutect2 tool are largely unfiltered. The raw variants need to be filtered using a separate tool: `FilterMutectMutations`. In order to optimally use this tool, we need to provide a contamination file. The tool that generates the contamination file is itself dependent on read pileup information around known variants. The workflow to generate these files and perform filtering is as follows:

5. Get pileup information for the samples at sites of known mutations for both the tumor and the normal files.

```
gatk-launch GetPileupSummaries \
   -I Tumor_01_recal.bam \
   -O Tumor_01.pileups.table \
    --variant reference/af-only-gnomad.raw.sites.b37.vcf. \--
java-options '-Xmx8g -Djava.io.tmpdir=tmp'
gatk-launch GetPileupSummaries \
   -I Normal_01_recal.bam \
   -O Normal_01.pileups.table \
    --variant reference/af-only-gnomad.raw.sites.b37.vcf. \--
java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

6. Estimate contamination. The `CalculateContamination` tool estimates the proportion of reads originating from other samples.

```
gatk-launch CalculateContamination \
   -I Tumor_01.pileups.table \
   -O Tumor_Normal_01.contamination.table \
   --matched-normal Normal_01.pileups.table \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

7. Apply a first (main) set of filters to variant calls:

```
gatk-launch FilterMutectCalls \
   --variant Tumor_Normal_01.Mutect2.raw.vcf.gz \
   -O Tumor_Normal_01.Mutect2.oncefiltered.vcf.gz \
   --contamination-table Tumor_Normal_01.contamination.table \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

The FILTER column in a VCF file is populated with results of the `FilterMutectCalls` tool, which currently consists of a set of 14 filters. See the Broad GATK documentation for a description of

the various filters. Passing variants will be labeled with a PASS in the FILTER field, and variants flagged by any of the filters will be retained but with the FILTER field populated with the list of filters for which the variant failed (*see* **Note 26**).

8. (Optional) Sequencing quality measures can be generated to additionally filter variants using the tool `FilterByOrientationBias`. To do this, the Picard tool `CollectSequencingArtifactMetrics` is first run to measure a set of sequencing errors. These errors are divided into two categories: pre-adapter and bait-bias artifacts. Only the pre-adapter error measurements will be used for variant filtering. The following command will produce four files: summary and detailed metrics for each of the two categories of error mentioned above.

```
gatk-launch CollectSequencingArtifactMetrics \-I Tumor_01_re-
cal.bam \
   -O Tumor_01.seqartifactmetrics \
   -R reference/ucsc.hg19.fasta \
   --INTERVALS exome_capture.target.intervals \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

9. Apply second pass filter to mark sequencing artifacts.

```
gatk-launch FilterByOrientationBias \
   --variant Tumor_Normal_01.Mutect2.oncefiltered.vcf.gz \
  --pre-adapter-detail-file Tumor_01.seqartifactmetrics.pre_-
adapter_detail_metrics \
   -O Tumor_Normal_01.Mutect2.twicefiltered.vcf.gz \
   --java-options '-Xmx8g -Djava.io.tmpdir=tmp'
```

Note that Mutect2 filter commands do not actually exclude the variant from the VCF, but instead mark the variant as suspect by adding a value to the VCF filter field. In effect, this documents that a candidate variant was considered by Mutect2, but ultimately deprecated as a false positive result. By marking variants as filtered Mutect2 allows you to review why a specific variant was filtered and to adjust the filtering parameters as necessary.

It is often desirable to focus on those variants that passed all filters. The following command excludes any variant which failed any filter (*see* **Note 27**).

```
awk '/^#/ {print $0; next} $7=="PASS" {print $0}' \
   Tumor_Normal_01.Mutect2.twicefiltered.vcf.gz \
   > sample01.T_v_N.vcf
```

At this point we have a set of filtered somatic variant calls that are ready for annotation.

### 3.5 Variant Annotation

Having determined the chromosome and position of putative variants, it is possible to apply gene models to annotate the variant loci with a gene, predict the impact of the specific variant on the transcription or translation of a gene, and, in general, begin to put the variant loci into a biological context. Biologically useful annotation is complex because it integrates several sources of large, complex, and dynamic data (e.g., gene and transcript models, structural and functional impact prediction, and third party annotations like dbSNP). Biological annotations are also often nuanced because the annotated result is sometimes ambiguous (e.g., the same locus may resolve to more than one gene or may have different impacts for different transcripts). Fortunately, there are many excellent open source tools that expedite annotation and analysis of a VCF file including VCFtools [33], SnpEff/SnpSift [34, 35], Variant Effect Preditor (VEP) [36], and ANNOVAR [37] as well as commercial tools such as SVS/VarSeq [38]. This workflow will focus on a small subset of open source tools to illustrate a simple approach to adding a common set of useful annotations:

- vcftools to add flanking sequence information.
- SnpEff to add gene name and id and structural/functional predictions.
- SnpSift to add population allele frequencies, and curated effect predictions from dbNSFP.

1. Add flanking sequence using VCFtools. VCFtools is suite of programs that expedite the manipulation and transformation of VCF files. The suite is composed of a core library and a set of Perl modules and scripts; we will focus on a single script, `fill-fs`, to add the flanking genomic sequence around the variant locus (*see* **Note 28**). The flanking sequence is necessary when designing primers (for orthogonal confirmation of variant) and flanking sequence is sometimes useful for ad-hoc/custom analysis of regional sequence patterns proximal to the variant.

   The `fill-fs` command below adds a new flanking sequence field (FS) into the existing INFO fields. The flanking sequence will show the 10 nucleotides on either side of the variant. Note that because the Variant Call Format standard is flexible, expressive, and broadly adopted, fill-fs, like most VCF annotation programs, accepts a VCF file as input and emits a VCF file as an output, appending annotation information to the existing variant lines as appropriate. This VCF-in, VCF-out convention enables one to chain smaller annotation steps together in an iterative series of commands.

```
scripts/fill-fs --refseq reference/ucsc.hg19.fasta \
   --length 10 \
   sample01.T_v_N.vcf \
```

```
  > sample01.T_v_N.annotated.flanking_sequence.vcf
```

The command below shows the first variant in the VCF; the excerpted output immediately below shows the new FS field:

```
egrep -m 1 '^[^#]' sample01.T_v_N.annotated.flanking_sequence.
vcf
chr1    14513    . G A ... FS=CAGGCAGACA[G/A]AAGTCCCCGC...
```

2. Add basic annotation and impact predictions with SnpEff. SnpEff interprets a VCF file against its database of features and emits a VCF annotated with information on the affected genes/transcripts along with predictions on the impact of each variant [34, 35]. The framework maintains a library of over 20,000 gene databases (multiple models, multiple versions, and many organisms). SnpEff requires you to choose a gene model database for annotation (e.g., Ensembl or UCSC) and SnpEff can reliably interpret Ensembl or UCSC chromosome names, so either hg19 or GRCh37 are valid database options for the work of this chapter (*see* **Note 29**).

This command lists the available SnpEff databases for *Homo sapiens*:

```
snpEff databases | awk '/Homo_sapiens/ {print $2, $1}'
Homo_sapiens GRCh37.75
Homo_sapiens GRCh38.86
Homo_sapiens hg19
Homo_sapiens hg19kg
Homo_sapiens hg38
Homo_sapiens hg38kg
Homo_sapiens testHg19ChrM
```

(Note that the databases are constantly updated, so versions may vary slightly from above.)

The SnpEff command below creates an annotated VCF (*see* **Notes 30** and **31**). It is important that your major genome version (e.g., hg19) matches the genome build you used in alignment.

```
snpeff -Xmx4g \
   -v \
   -stats annotated.snpeff_stats.html \
   hg19 \
   sample01.T_v_N.annotated.flanking_sequence.vcf \
   > sample01.T_v_N.annotated.snpeff.vcf
```

SnpEff typically adds one to three new INFO fields for each variant line:

- LOF: present only if the variant is predicted to cause Loss Of Function.
- NMD: present if the variant would result in Nonsense Mediated Decay.
- ANN: effect annotation (always present).

The ANN INFO field lists one or more predicted effects for the variant. A variant can have multiple effects because (a) there is more than one gene at the variant locus, (b) there is more than one transcript for a gene at the variant locus, (c) there is more than one alternate allele at the locus. SnpEff ranks and sorts the list of annotation effects from most deleterious (e.g., frameshift of a protein-coding gene leading to LOF) to least deleterious (e.g., intronic variant). The fact that each variant locus often has more than one effect complicates interpretation; a common simplification is to transform the VCF so that each line has exactly one effect. To make this transformation one has to choose between duplicating a single locus into a locus for each effect or filtering to a single effect per locus. In this example, we will use a script provided by SnpEff to filter out all but the top effect (i.e., we retain the most deleterious effect) (*see* **Note 32**):

```
cat samplet1.T_v_N.annotated.snpeff.vcf \
   | scripts/vcfAnnFirst.py \
   > sample01.T_v_N.annotated.top_effect.vcf
```

Note that each annotation effect is composed of structured sub-fields such as "annotation" (a sequence variant term from the Sequence Ontology database [39]), "gene name," "HGVS notation," and other information (*see* **Notes 33** and **34**).

3. (optional) Add dbNSFP info using SnpSift. SnpSift is a companion program of SnpEff that enables filtering, annotation, and other analysis/transformations of VCF files. Here, we use SnpSift to interpret the dbNSFP annotation database. dbNSFP (DataBase of NonSynonymous Functional Predictions) is an aggregation of human, single-nucleotide variant annotations; it includes variant population frequencies from several large cohorts, predicted functional impacts, conservation scores, and feature id mappings from an array of independent source databases [40, 41]. Adding dbNSFP annotations is a simple way to integrate more biological context into the variant analysis and adds important evidence to help interpret the significance of variant patterns in your data. Note that SnpEff adds basic impact prediction for every variant line, whereas SnpSift dbNSFP will only add annotation for variant loci found in the dbNSFP database.

The following command adds variant population frequencies from several cohorts, impacts prediction from ClinVar [42], and the Ensembl protein id (*see* **Note 35**). This command assumes you have installed the dbNSFP database as detailed above (*see* Subheading 3.1.3).

```
snpsift -Xmx4g \
   dbnsfp \
   -db reference/dbNSFP3.5a_hg19.txt.gz \
   -f gnomAD_exomes_AF,ExAC_AF,clinvar_clnsig,Ensembl_protei-
nid \
   -v sample01.T_v_N.annotated.top_effect.vcf \
    > sample01.T_v_N.annotated.dbnsfp.vcf
```

4. Convert VCF to a tab-separated text file. The Variant Call Format standard is both expressive and flexible; this is largely due to how the VCF structures INFO and FORMAT fields to allow varying numbers and sets of values for each variant line. That said, if you plan to do custom visualization or analysis of the variant data, it is often useful to transform a structured VCF file into a simpler, tabular text file. The commands below present two alternative ways to transform VCFs into tab-separated files that can be used independently or together.

SnpSift's `extractFields` command expands a subset of INFO ANN annotation subfields into discrete columns.

```
snpsift extractFields \
   sample01.T_v_N.annotated.dbnsfp.vcf \
   CHROM POS REF ALT \
   ANN[0].EFFECT ANN[0].IMPACT \
   ANN[0].GENE ANN[0].GENEID \
   ANN[0].HGVS_C \
   > sample0-1.T_v_N.annotated.extractFields.tsv
```

Compared to SnpSift `extractFields`, the Jacquard `expand` utility [43] has fewer options and cannot expand the ANN field into discrete fields; however, it can expand VCF FORMAT values into separate sample matrices for each FORMAT field and it produces a text glossary based on the INFO/FORMAT VCF metaheaders.

```
jacquard expand \
   sample01.T_v_N.annotated.dbnsfp.vcf \
   sample01.T_v_N.annotated.jacquard.tsv
```

Since the loci from the two tab-separated files above match line for line, they can be combined:

```
paste sample01.T_v_N.annotated.extractFields.tsv \
   sample01.T_v_N.annotated.jacquard.tsv \
   > sample01.T_v_N.annotated.combined.tsv
```

We now have fully filtered and annotated data files listing the variants detected in the experiment.

## 3.6 Supplement: Alternative Variant Detection Workflow

We provide here an alternate variant calling workflow, one that uses fully release production-level versions of the tools in the GATK toolkit, as well as an alternative variant caller: VarScan2 Somatic. VarScan2 is an intuitive variant caller that implements a straightforward Fisher's Exact Test based scoring metric to detect the presence of variants from read pileup data from `samtools mpileup`. In addition to somatic variants, the somatic version of the caller will by default generate lists of germline and loss-of-heterozygosity (LOH) variants. Variants are then filtered using additional tools packaged with the main caller. Because this variant caller does not perform local realignment, it is good practice to implement a realignment procedure around regions of known indels using the tools provided in legacy versions of the GATK.

It can be beneficial to utilize more than one variant detection algorithm when calling mutations, either in a comparative manner or in combination as an ensemble or consensus method [44–46]. Properly tuned, most callers can perform similarly, but there will typically be classes of variants more easily identified by one tool over another.

Overall, this supplementary section is meant to replace Subheading 3.2 through Subheading 3.4 in the main workflow. Once a final filtered variant set is produced, the same annotation steps may be performed starting with Subheading 3.5.

### 3.6.1 Supplemental Setup

The tools used in this supplemental workflow are listed in Table 2.

1. Set up an environment variable for the main folder. This may be done using the "export" command on most linux platforms:

```
export WES=/path/to/project/root
```

2. Create the project home folder structure:

```
cd $WES
mkdir -p input reference scripts tmp
```

3. Download and install the Conda package manager. Miniconda should be fine, and Python2 is preferred for compatibility with the annotation software.
4. Configure Conda channels.

**Table 2**
**Tools for somatic variant identification and annotation using VarScan**

| Package (version) | Tool | Function as used in this document |
|---|---|---|
| bam-readcount (0.8.0) | `bam-readcount` | Generate read depth and metrics at single-nucleotide positions |
| BWA (0.7.15) | `index`<br>`Mem` | Index database sequences in the FASTA format<br>Align reads to reference genome |
| Conda (4.3.21) | `config` | Install and manage packages and environment |
| FastQC (0.11.5) | `fastqc` | Assess sequencing read quality |
| GATK (3.8) | `BaseRecalibrator`<br>`CreateSequenceDictionary`<br>`DepthOfCoverage`<br>`FixMateInformation`<br>`IndelRealigner`<br>`PrintReads`<br>`RealignerTargetCreator` | Detect systematic errors in base quality scores<br>Create a sequence dictionary for a reference sequence<br>Process BAM files to determine coverage at different levels of partitioning and aggregation<br>Verify mate-pair information between mates and fix if needed<br>Perform local realignment of reads around indels<br>Apply base quality score recalibration<br>Define intervals to target for local realignment |
| Jacquard (0.42) | `expand` | Expand a VCF file into a TXT |
| Picard (2.17.6) | `MarkDuplicates`<br>`ReorderSam`<br>`SortSam` | Locate and tag duplicate reads in an alignment file<br>Reorder reads in an alignment file to match the contig ordering in a provided reference file<br>Sort, compress and index alignment |
| SAMtools (1.2) | `faidx`<br>`flagstat`<br>`mpileup` | Index reference sequence in the FASTA format<br>Report alignment statistics<br>Generate pileup from BAM files |
| SnpEff (4.3T) | `database`<br>`snpeff` | List all supported databases<br>Annotate genomic variants and predict functional effect |
| SnpSift (4.3T) | `dbnsfp`<br>`extractFields` | Annotate genomic variants using dbNSFP<br>Extract fields from a VCF file to a TXT |
| Trimmomatic (0.36) | `trimmomatic-0.36.jar` | Trim reads for quality |

(continued)

**Table 2**
**(continued)**

| Package (version) | Tool | Function as used in this document |
|---|---|---|
| VarScan (2.4.3) | `fpfilter` | Filter false positive variant calling results |
| | `processSomatic` | Process VarScan output by somatic status and confidence |
| | `somatic` | Call variants and identify somatic status using pileup files from a matched tumor-normal pair |
| VCFtools (0.1.15) | `fill-fs` | Annotate VCF with flanking sequence |

```
conda config --add channels r
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

5. Create a new Conda environment:

```
conda create --name altwes \
   bwa fastqc gatk R samtools trimmomatic picard bam-readcount
varscan
```

6. Activate the conda environment:

```
source activate altwes
```

7. Download the GATK version 3.8 package into the scripts folde. This should be a zipped archive:

```
cd $WES/scripts
wget -r -np -nd \
   -O gatk3.tar.bz2 \
'https://software.broadinstitute.org/gatk/download/auth?pack-
age=GATK-archive&version=3.8-0-ge9d806836'
bunzip2 gatk3.tar.bz2
   tar -xf gatk3.tar
```

8. Link the GATK jar file to the `gatk` command in Conda:

```
gatk-register $WES/scripts/GenomeAnalysisTK-3.8-0-ge9d806836/
GenomeAnalysisTK.jar
```

9. Download the Broad GATK Base Quality Score Recalibration plotting script:

```
cd $WES/scripts
wget -np -nd \
https://github.com/broadgsa/gatk/blob/master/public/gatk-engine/src/main/resources/org/broadinstitute/gatk/engine/recalibration/BQSR.R
```

10. Follow main workflow preprocessing steps in Subheadings "Install Software for Annotation" and 3.1.3 to finish software and reference file installation.

### 3.6.2 Preprocessing A

An outline of this supplemental workflow is shown in Fig. 2.

1. Follow Subheading 3.2, **steps 1–3** in the main workflow to produce alignment BAM files for the Tumor_01 and Normal_01 samples, which are equivalent. The following workflow starts at the BAM file sorting step.
2. Sort, compress, and index the alignment file. This step will result in a single, sorted, index BAM file.

```
picard SortSam \
   INPUT=Tumor_01.sam \
   OUTPUT=Tumor_01.bam \
   SORT_ORDER=coordinate \
   CREATE_INDEX=true \
   VALIDATION_STRINGENCY=LENIENT
```

(repeat for the normal sample)

Note that once BAM files have been created, the Tumor_01.sam and Normal_01.sam files can be removed. They can be large and it may be desirable to save space. SAM files can be regenerated from their BAM file equivalents at any time.

3. Mark duplicate reads in the alignment file. For any set of paired reads that have precisely the same coordinates when aligned to the reference genome, which are often PCR duplicates, only the highest-scoring reads are left unmarked (*see* **Note 17**).

```
picard MarkDuplicates \
     INPUT=Tumor_01.bam \
     OUTPUT=Tumor_01_mkdp.bam \
     REMOVE_DUPLICATES=false \
     METRICS_FILE=Tumor_01_rmdp_metrics.txt \
     CREATE_INDEX=true \
     VALIDATION_STRINGENCY=LENIENT
```

(repeat for the normal sample)

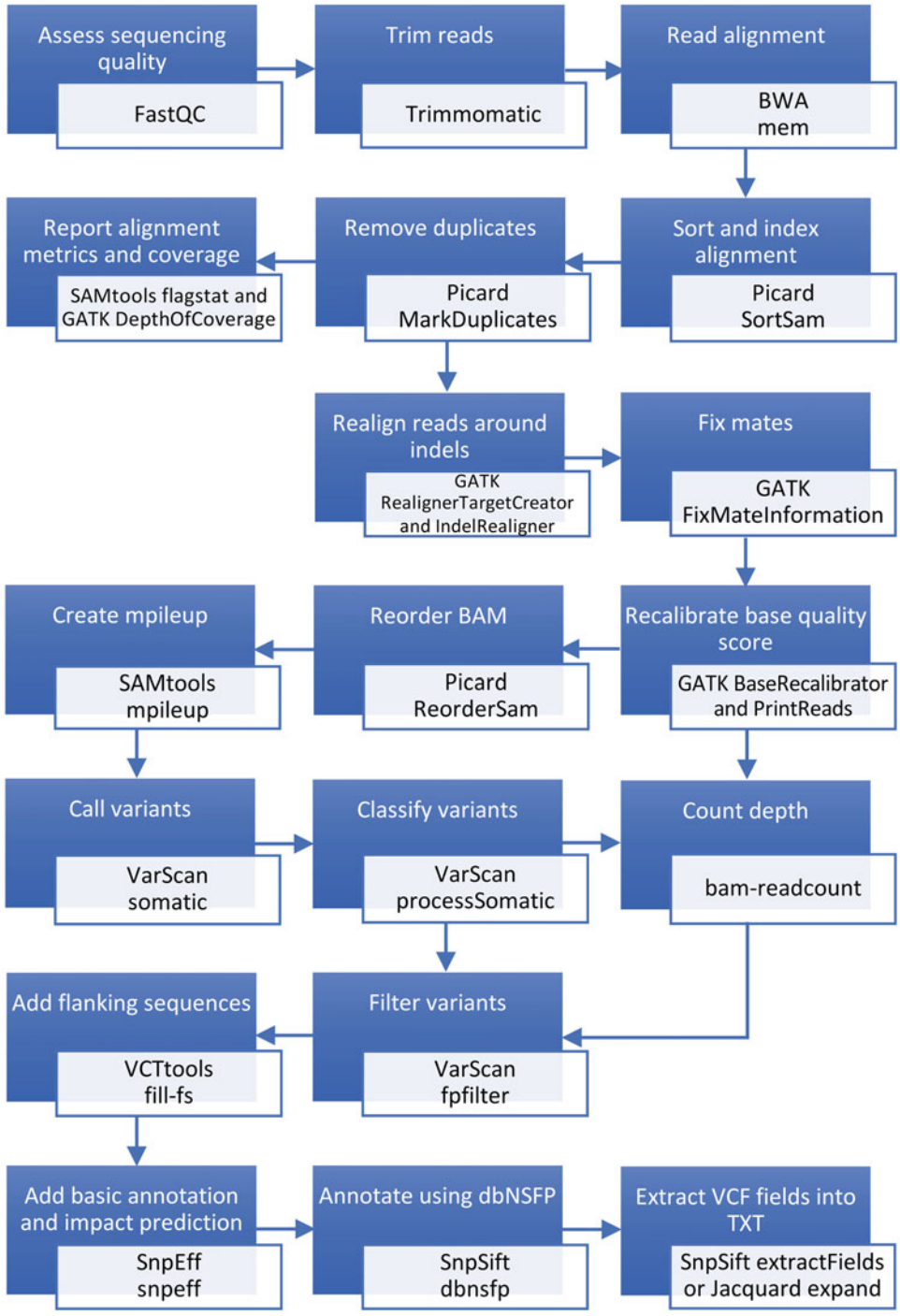


**Fig. 2** Workflow of somatic variant identification and annotation using GATK3 and VarScan

4. Generate basic metrics on the de-duplicated BAM files (optional). The `samtools flagstat` tool will assemble overall read counts, pairing, mapping, and deduplication metrics on

BAM files. The `flagstat` tool is described at the following location: http://www.htslib.org/doc/samtools.html.

```
samtools flagstat Tumor_01.bam > Tumor_01.bam.flagstat
```

(repeat for the normal sample)

5. Generate overall coverage data (optional). Rather than using Picard's `CalculateHSMetrics`, here we provide a profile using the GATK `DepthOfCoverage` tool which is available in GATK3. Several files will be produced, but the `Tumor_coverage.sample_summary` file will contain mean, median, and quantile read depths across the regions listed in a provided bed file, as well as the fraction of bases in the regions that achieve the depths specified by the `-ct` arguments.

```
gatk -Xmx8g -Djava.io.tmpdir=tmp \
   -T DepthOfCoverage \
   -R reference/ucsc.hg19.fasta \
   -I Tumor_01.bam \
   -o Tumor_01_coverage
   -L input/exome_capture.target.bed \
   -ct 1 -ct 20 -ct 50 -ct 100
```

### 3.6.3 Preprocessing B: Indel Realignment and Base Quality Score Recalibration

The equivalent BQSR preprocessing step described in the main workflow is designed for use with a variant caller such as Mutect2 or FreeBayes that utilizes a local haplotype assembly step. In earlier GATK workflows which utilized UnifiedGenotyper, BQSR was preceded by a local realignment step around regions of known Indels. Indel realignment attempted to minimize the number of bases that are mismatched to the reference genome in regions around insertions and deletions, regions in which an alignment algorithm is most likely to generate misaligned bases in single reads. This step is recommended when using a variant caller that does not perform a local realignment.

#### Local Realignment Around Indels

1. Create a target intervals list:

```
gatk -Xmx8g -Djava.io.tmpdir=tmp \
   -T RealignerTargetCreator \
   -R reference/ucsc.hg19.fasta \
   -I Tumor_01_mkdp.bam \
   -o Tumor_01_target.list \
   -known reference/1000G_phase1.indels.hg19.vcf \
  -known reference/Mills_and_1000G_gold_standard.indels.hg19.
vcf
```

(repeat for the normal sample)

2. Actually perform local realignment on reads overlapping the target intervals:

```
gatk -Xmx8g -Djava.io.tmpdir=tmp \
   -T IndelRealigner \
   -R reference/ucsc.hg19.fasta \
   -I Tumor_01_rmdp.bam \
   -o Tumor_01_realign.bam \
   -targetIntervals Tumor_01_target.list \
   -known reference/1000G_phase1.indels.hg19.vcf \
  -known reference/Mills_and_1000G_gold_standard.indels.hg19.
vcf
```

(repeat for the normal sample)

3. Ensure that all mate-pair information is in sync between each read and its mate pair (*see* **Note 36**).

```
gatk -Xmx8g -Djava.io.tmpdir=tmp \
   -T FixMateInformation \
   INPUT=Tumor_01_realign.bam \
   OUTPUT=Tumor_01_fixmate.bam \
   SO=coordinate \
   VALIDATION_STRINGENCY=LENIENT \
   CREATE_INDEX=true
```

(repeat for the normal sample)

Base Quality Score Recalibration

4. Build recalibration model

```
gatk -Xmx8g -Djava.io.tmpdir=tmp  \
   -T BaseRecalibrator \
   -I Tumor_01_fixmate.bam \
   -R reference/ucsc.hg19.fasta \
    -knownSites:dsbsnp_137,VCF reference/dbsnp_138.hg19.exclu-
ding_sites_after_129.vcf.gz \
     -knownSites:Mills_indels,VCF  reference/Mills_and_1000G_-
gold_standard.indels.hg19.vcf \
  -knownSites:1000G_indels,VCF reference/1000G_phase1.indels.
hg19.vcf \
   -o Tumor_01_bqsr.table
```

(repeat for the normal sample)

5. Perform recalibration

```
gatk -Xmx8g -Djava.io.tmpdir=tmp \
   -T PrintReads \
   -R reference/ucsc.hg19.fasta \
   -I Tumor_01_rmdp.bam \
   -BQSR Tumor_01_bqsr.table \
   -o Tumor_recal.bam
```

(repeat for normal sample)

At this point, we have recalibrated alignment (BAM) files ready for variant calling.

### 3.6.4 Variant Calling and Filtering

We'll first prepare the files for running VarScan by insuring the reads are ordered correctly and by generating `samtools` pileups.

1. reorder BAM:

```
picard ReorderSam \
   REFERENCE=reference/ucsc.hg19.fasta \
   CREATE_INDEX=true \
   INPUT=Tumor_01_recal.bam \
   OUTPUT=Tumor_01_reordered.bam
```

(repeat for normal sample)

2. Generate `mpileup` read depth information on all locations in the genome at single-base resolution (*see* **Note 37**).

```
samtools mpileup -q 1 -d 5000 \
   -f reference/ucsc.hg19.fasta \
   Tumor_01_reordered.bam  > Tumor_01.pileup
```

(repeat for normal sample)

3. Run base VarScan somatic on the tumor/normal sample pair. This command will result in two files in VCF format, one for SNPs and one for Indels (*see* **Note 38**).

```
varscan somatic Normal_01.pileup Tumor_01.pileup \
   Tumor_vs_Normal_01.varscan \
   --min-coverage-normal 8 \
   --min-coverage-tumor 6 \
   --min-var-freq 0.10 \
   --min-freq-for-hom 0.75 \
   --tumor-purity 0.95 \
   --p-value 0.99 \
   --somatic-p-value 0.05 \
   --strand-filter 1 \
   --output-vcf
```

The parameters that are specified reflect default values, but attention should be paid to them since they're not optimal for all situations. Of particular note is the `-min-var-freq` parameter, which is a lower limit on the alternate allele frequency at which the tool will call a heterozygous (0/1) mutation. See the VarScan User's Manual for more information on these parameters: http://varscan.sourceforge.net/using-varscan.html.

4. Process and classify bulk variants using VarScan2 `processSomatic`. This step will classify the variants generated in the previous step into three categories (somatic, germline, and LOH), and additionally produce high-confident subsets of each of these categories. The command will be run separately on the two SNP and INDEL VCF files, producing a total of 12 VCF files in all.

```
varscan processSomatic Tumor_vs_Normal_01.varscan.snp.vcf \
   --min-tumor-freq 0.10 \
   --max-normal-freq 0.05 \
   --p-value 0.05
varscan processSomatic Tumor_vs_Normal_01.varscan.indel.vcf \
   --min-tumor-freq 0.10 \
   --max-normal-freq 0.05 \
   --p-value 0.05
```

5. Extract variant coordinates from VCF files.

```
awk 'BEGIN {OFS="\t"} {if (!/^#/) { isDel=(length($4) > length
($5)) ? 1 : 0; print $1,($2+isDel),($2+isDel); }}' Tumor_vs_-
Normal.varscan.snp.Somatic.hc.vcf > Tumor_vs_Normal.varscan.
snp.Somatic.hc.var
awk 'BEGIN {OFS="\t"} {if (!/^#/) { isDel=(length($4) > length
($5)) ? 1 : 0; print $1,($2+isDel),($2+isDel); }}' Tumor_vs_-
Normal.varscan.indel.Somatic.hc.vcf > Tumor_vs_Normal.vars-
can.indel.Somatic.hc.var
```

6. Generate readcount files for the high-confident somatic variants (*see* **Note 39**).

```
bam-readcount -q10 -b20 -w1 \
   -l Tumor_vs_Normal.varscan.snp.Somatic.hc.var \
   -f reference/ucsc.hg19.fasta  \
   Tumor_reordered.bam \
   > Tumor_vs_Normal.varscan.snp.Somatic.hc.readcount
bam-readcount -q10 -b20 -w1 \
   -l Tumor_vs_Normal.varscan.indel.Somatic.hc.var\
   -f reference/ucsc.hg19.fasta\
   Tumor_reordered.bam\
   > Tumor_vs_Normal.varscan.indel.Somatic.hc.readcount
```

7. Variants can now be filtered using the VarScan `fpfilter` tool. This tool has a number of customizable parameters which may be tuned depending on the experiment and the tolerance for false positives. The default parameters for this tool are based on read lengths of at least 100bp and eliminate reads with mapping qualities below a threshold as well as reads detecting the variant close to the ends of reads from contributing to a variant call. One particular variable of note is the `--min-var-count` parameter, which specifies the minimum number of alternate allele supporting reads (default 4). The full list of parameters may be found in the VarScan User's Manual, or by running the command: `varscan fpfilter --help` (*see* **Note 40**).

   The following commands will each generate a VCF file containing the somatic variants that pass the specified filters.

```
varscan fpfilter Tumor_vs_Normal.varscan.snp.Somatic.hc.vcf \
   Tumor_vs_Normal.varscan.Somatic.snp.hc.readcount \
    --output-file Tumor_vs_Normal.varscan.Somatic.snp.hc.fil-
tered.vcfvarscan fpfilter Tumor_vs_Normal.varscan.indel.So-
matic.hc.vcf \
   Tumor_vs_Normal.varscan.Somatic.indel.hc.readcount \
  --output-file Tumor_vs_Normal.varscan.Somatic.indel.hc.fil-
tered.vcf
```

At this point we have a set of filtered somatic variant calls that are ready for annotation. Proceed with Subheading 3.5 Variant Annotation in the main workflow.

The workflows we have outlined here result in an annotated list of variant calls. We largely utilized default parameters for many of the steps in these workflows, but that choice is not always appropriate. There are a lot of potential decision points when running these steps, many of which will have an impact on the final set of calls. It is best to become familiar with the parameter options and details of the main tools used in any workflow. If gold standard datasets are available, benchmarking any parameter changes or tool version upgrades using those datasets will help avoid problems and surprises.

## 4 Notes

1. As of the time of this writing, a recent survey of variant calling tools may be found in the ThousandVariantCallersRepo (1KVCP): https://github.com/deaconjs/ThousandVariantCallersRepo/wiki/SNV.
2. The exome library capture platform and prep protocol used to prepare the sequencing library may influence the choice of

sequencing read length. The protocol may, for example, contain a size selection step which filters DNA fragments for sequencing to those of length 150–400bp. Also, the baits or probes in the capture method may be of specific sizes that perform more optimally with DNA of specific fragment lengths. Generating reads data that are, for example, longer than those recommended by the exome capture protocol may result in excessive read-through into adapter, or excessive overlapping of paired reads.

3. The most frequently utilized gene models for most commercial exome capture platforms are RefSeq and ENSEMBL, although other data sets such as CCDS, Gencode, VEGA, SNP, and CytoBand may be used. There are differences between the commercial platforms in the degree to which they target untranslated regions (UTRs) of each gene which can have a significant impact on the size of the target. It is important to make sure that the reference genome version used in the bioinformatics workflow matches the version used by the vendor in their probe design.
4. A description of BED format may be found here: https://genome.ucsc.edu/FAQ/FAQformat.html#format1
5. A VCF file may be indexed using the `IndexFeatureFile` tool in the GATK package using the following command. *See* also **Note 12**.

```
gatk-launch IndexFeatureFile --feature-file my.vcf
```

6. More information on the adapter files may be found on the main tool website: http://www.usadellab.org/cms/index.php?page=trimmomatic.
7. It is necessary for the data resource files chromosome naming convention to match the one used by the reference genome. In the reference genome FASTA file, each chromosome such as Chromosome 1 may be listed as either "chr1" or simply "1," depending on the specific source from which the file was downloaded. The version of the reference genome in the GATK toolkit follows the "chr1" naming convention. However, the af-only-gnomad.raw.sites.b37.vcf file that is currently provided in the /bundle/beta/Mutect2/ resource bundle directory follows the "1" convention. If this has not changed at the time of reading, the af-only-gnomad.raw.sites.b37.vcf file will need to be modified by prepending the "chr" string to every variant line in the file. This can be done as follows:

```
mv af-only-gnomad.raw.sites.b37.vcf.gz \
   af-only-gnomad.raw.sites.b37.vcf.original.gz
```

```
gzip -d -c af-only-gnomad.raw.sites.b37.vcf.original.gz  \
   | awk '/^#/ {print $0;next} $1== "MT" \
   {$1=="M"}{print "chr" $0}' \
   | bgzip > af-only-gnomad.raw.sites.b37.vcf.gz \
   && tabix af-only-gnomad.raw.sites.b37.vcf.gz
```

Note that the strong preference is always to use resources whose chromosome naming conventions match; with that in mind, we look forward to Broad aligning this file's format with the rest of their bundle rending the transformation above obsolete.

8. In a hybrid-capture assay, there are two types of region files that are often referred to: a "target" or "primary" region and a "bait" or "capture" region. The former "target" region refers to regions that the assay is targeting, in the case of WES the specific exonic regions of genes in the genome. The "bait" region refers to the overall footprint of the overlapping capture probe sequences. These files may be specified in BED format, however some tools (particularly the ones based on Picard) require a slightly different "interval" format, which is basically a BED format with an additional SAM-like header section.
9. By default, GATK and other tools will utilize a server's /tmp folder space when executing commands. This folder space may have a limited allocated size which can fill up, depending on dataset size, the number of concurrent jobs, and other users. Specifying a local location when executing tool commands mitigates the likelihood of a job crash due to insufficient resources.
10. See the VCFtools website for a full description of the library of utility programs and also alternative ways to install VCFtools: https://vcftools.github.io/perl_module.html
11. We use Python and bash scripts to find the absolute path of the Conda-installed SnpEff scripts dir. This Python approach is preferred over standard Unix `readlink` because the macOS uses a non-standard implementation of `readlink`.
12. This is the first command in this workflow to utilize the GATK toolkit. Unlike earlier versions in which each command was specified directly via the java JAR file, the GATK library now includes a command dispatch framework that is invoked via the `gatk-launch` or simply the `gatk` command (depending on the package).
13. The initial download is currently ~17Gb; expansion of intermediate files requires ~170Gb; the final database is ~40Gb. The scripts are simple and the process is largely unattended, but is reasonable to budget several hours to download and several more hours of compute time to prepare the files. This example assumes you are working with the latest version at the time of this writing, dbNSFP3.5a (academic version), and

using the hg19 genome assembly. dbNSFP aggregates many variant annotation sources together and several data sources have licensing restrictions on commercial use. To accommodate these restrictions, dbNSFP publishes two branches of the database: dbNSFP3.5a for academic use and a slightly reduced dbNSFP3.5c for commercial use. For more info see dbNSFP website: https://sites.google.com/site/jpopgen/dbNSFP

14. These instructions are adapted from the dbNSFP download approach outlined on the SnpSift website: http://snpeff.sourceforge.net/SnpSift.html#dbNSFP
15. This assumes you are using dbNSFP version 3.5a; adjust the `version="3.5a"` command as necessary.
16. A description of the individual tools in a FastQC report may be found here: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/. A handy optional tool to use in conjunction with FastQC is MultiQC [23].
17. Adapter sequence most typically occurs at the 3' end of reads, in cases where the insert size of a DNA fragment is short enough to permit sequencing to extend beyond the target sequence and into adapter. Most library prep protocols for exome sequencing include a size selection step that targets DNA fragments of a specified length, e.g., 320bp.
18. Exome sequencing is almost always based on a capture or affinity-based enrichment strategy, as opposed to an amplicon-based enrichment strategy often found in smaller targeted sequencing experiments. When the resulting sequencing reads from capture-based strategy are aligned to the reference genome, they result in a more continuous, normal-shaped pileup distribution. This distribution is amenable to deduplication, which attempts to mitigate PCR-based artifacts. Duplicates that are marked are retained in the alignment files, but most variant calling algorithms will ignore them. If using a variant caller which does not ignore duplicates, it is probably desirable to change the `--REMOVE_DUPLICATES` flag to `true`. Note, there is a new tool in the latest GATK4 suite called MarkDuplicatesGATK, which may provide improvement over MarkDuplicates, but it is currently listed as experimental.
19. If only one region file is available, use the same file for both TARGET and BAIT intervals.
20. In older versions of the GATK Best Practices workflow, BQSR was preceded by a local realignment step around regions of known indels. Indel realignment is a superfluous step, however, if one is using a variant caller that performs local haplotype assembly, such as Mutect2, HaplotypeCaller, or FreeBayes. In fact, the tools to perform local realignment are no longer

provided in the current (GATK4) library. If one is working with a current variant caller that does not perform local realignment and assembly (e.g., VarScan2), Indel realignment is recommended. Please see the appendix for an example of a workflow that utilizes indel realignment.

21. The `AnalyzeCovariates` tool, when invoked using default parameters, will attempt to launch an internal R script called BQSR.R. This tool appears to make some assumptions about the R language environment of the computer on which it is running, and we have often seen it fail. The version of the command we are specifying here produces an intermediate data file in CSV format, circumventing the need for the `AnalyzeCovariates` tool to launch Rscript internally. Once this file is produced, the BQSR.R script can be run on it directly as specified. The BQSR.R script may be found on the Broad GATK Github repository and should be easily found via a web search.
22. According to the Broad Institute, Mutect2 refers to the version of the tool provided in GATK4, whereas MuTect2 (with the first "T" capitalized) refers to earlier version provided in GATK3.
23. `gatk-launch --java-options '-Xmx8g -Djava.io.tmpdir=tmp_${f}_Mutect2' Mutect2 -R ${TOOLS}/ucsc.hg19.fasta -I Tumor_01_recal.bam -tumor Tumor_01 -L ${TOOLS}/exome_capture.bed -O Tumor_01.Mutect2.vcf.gz --germline-resource ${TOOLS}/af-only-gnomad.raw.sites.b37.pju.vcf --panel-of-normals Project_PoN.vcf.gz --af-of-alleles-not-in-resource .0000025`
24. Mutect2 in tumor-only mode is not supported by the Broad GATK team at the time of this writing and running in this mode is not generally recommended. For tumor-only variant calling, running a germline caller such as HaplotypeCaller, FreeBayes, or VarScan (germline) is currently a safer option. With these tools, variants will be detected but their somatic status cannot be distinguished from germline. If running Mutect2 without a matched normal is still a goal, know that some parameter tuning will almost certainly be necessary. By default, the germline risk that is estimated by Mutect2 for a variant can be overly stringent, resulting in the "germline_risk" filter being marked on the majority or all of somatic variants. In the command provided in Note 23, the `--af-of-alleles-not-in-resource` argument is provided to attempt to mitigate that effect, set to 1/400,000 based on the fact that this is WES using the GnomAD germline resource (see the Broad GATK online forum discussion #10157 for more information).

25. See the following document for information on the VCF file format specification: https://samtools.github.io/hts-specs/VCFv4.1.pdf.
26. The Broad GATK team has indicated that a Deep Learning based variant filtering tool is in the works, which may deprecate this form of variant calling.
27. Mutect2 marks variants that pass all filters with the filter value PASS; this is consistent with the VCF specification. That said, other variant callers approach filtering differently, and, depending on the variant caller, you may need to adjust the command listed in the text or you may not need to filter at all.
28. There are many more utility scripts in VCFtools; see the website for more information: https://vcftools.github.io/perl_module.html.
29. SnpEff documentation encourages the use of Ensembl (GRCh37) annotations over UCSC (hg19) because unlike UCSC, Ensembl employs precise and explicit version for each annotation build. We use hg19 for this example because the annotated VCF will be emitted with the same chromosome naming convention as the input VCF (hg19).
30. In addition to the annotated VCF, SnpEff also emits a comprehensive HTML report of the overall annotation results as well as a tab-separated text matrix of genes/transcript by effect type.
31. The first time you invoke this command, it will download and install the appropriate database of features; a human SnpEff database is roughly 700Mb so this may take a few minutes depending on your connectivity.
32. If you need a single effect per variant locus, but want to preserve all effects, this alternative transformation uses a different SnpEff script to duplicate the variant lines once for each effect. Naturally, this file will have more lines than the original annotated VCF.

```
cat sample01.T_v_N.annotated.snpeff.vcf \
  | perl scripts/vcfEffOnePerLine.pl \
  > sample01.T_v_N.annotated.one_eff_per_line.vcf
```

33. Like all other INFO fields, the structure of the ANN field is documented in a line the VCF metaheader:

```
egrep -m 1 \
  -e '^##INFO=<ID=ANN,' \
   sample01.T_v_N.annotated.top_effect.vcf##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele | Annotation | Annotation_Impact | Gene_Name | Gene_ID |
```

```
Feature_Type | Feature_ID | Transcript_BioType | Rank | HGVS.c
| HGVS.p | cDNA.pos / cDNA.length | CDS.pos / CDS.length | AA.
pos / AA.length | Distance | ERRORS / WARNINGS / INFO' ">
```

The command below joins the field labels above with field values of the effect for the first variant in the VCF:

```
paste -d : \
    <(egrep -m 1 -e '^##INFO=<ID=ANN,' \
            sample01.T_v_N.annotated.top_effect.vcf \
       | sed 's/.*Desc.*="Func.* annotations: .\(.*\). ">/\1/' \
        | sed 's/ | /|/g' \
        | tr '|' '\n') \
    <(egrep -m 1 -e '^[^#]' \
            sample01.T_v_N.annotated.top_effect.vcf \
        | cut -f8 \
        | sed 's/.*ANN=\(.*\)/\1/' \
        | sed 's/ | /|/g' \
        | tr '|' '\n') \
   | column -t -s ':'Allele                    A
Annotation                downstream_gene_variant
Annotation_Impact         MODIFIER
Gene_Name                 DDX11L1
Gene_ID                   DDX11L1
Feature_Type              transcript
Feature_ID                NR_046018.2
Transcript_BioType        pseudogene
Rank
HGVS.c                    n.*104G>A
HGVS.p
cDNA.pos / cDNA.length
CDS.pos / CDS.length
AA.pos / AA.length
Distance                  104
ERRORS / WARNINGS / INFO
```

34. For more details on interpreting the ANN field, see the snpeff website and this link in particular: http://snpeff.sourceforge.net/VCFannotationformat_v1.0.pdf. For more information on the ANN annotation subfield, see the Sequence Ontology Browser website: http://www.sequenceontology.org/browser/obob.cgi.
35. When SnpSift `dbnsfp` runs, it lists all the available dbNSFP fields in its console log. See the README file link on the dbNSFP website for a complete list and short definition of available annotation fields: https://sites.google.com/site/jpopgen/dbNSFP.

36. Running the Picard `FixMateInformation` step may be redundant; it is something that the GATK tools should take care of. Running this step is precautionary.
37. By default, `mpileup` will restrict the maximum read depth to 250 reads; here we raise that limit to a higher amount to accommodate the possibility of greater sequencing depth. We also limit the pileups to reads with mapping quality greater than 0 with the -q 1 parameter.
38. VarScan has two overall output modes: a mode that produces a native text format, and a mode that produces VCF files. We're outlining steps that use the VCF format, indicated by the `--output_vcf` parameter.
39. When extracting read depth information from the BAM files at the locations of the detected variants, the Tumor BAM file should be used when generating data for the Somatic variants, and the Normal BAM file for the Germline or LOH variants.
40. VarScan2 was used to participate in the ICGC-TCGA DREAM Genomic Mutation Calling Challenge, a competition based on identifying cancer-associated mutations and rearrangements in whole-genome sequencing (WGS) data. The `fpfilter` parameters found to be optimal for those data can be set by specifying the `--dream3-settings` parameter when running the tool. More information on this challenge may be found at the following URL: https://www.synapse.org/#!Synapse:syn312572/wiki/

## Acknowledgments

The authors would like to thank the institutions, developers, and documenters of the informatics tools used in this chapter's workflows. Genomics and disease research in general benefits hourly from the availability of tools such as Bioconda, BWA, GATK, HaplotypeCaller, Mutect2, Samtools, SNPEff , VarScan, and Vcftools, as well as public resources such as ClinVar and GnomAD.

### References

1. Karapetis CS, Khambata-Ford S, Jonker DJ et al (2008) K-ras mutations and benefit from cetuximab in advanced colorectal cancer. N Engl J Med 359:1757–1765
2. DePristo MA, Banks E, Poplin R et al (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet 43:491–498
3. McKenna A, Hanna M, Banks E et al (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res 20:1297–1303
4. Hwang S, Kim E, Lee I et al (2015) Systematic comparison of variant calling pipelines using gold standard personal exome variants. Sci Rep 5:17875
5. Cornish A, Guda C (2015) A comparison of variant calling pipelines using genome in a bottle as a reference. Biomed Res Int 2015:456479

6. Roberts ND, Kortschak RD, Parker WT et al (2013) A comparative analysis of algorithms for somatic SNV detection in cancer. Bioinformatics 29:2223–2230
7. Wang Q, Jia P, Li F et al (2013) Detecting somatic point mutations in cancer genome sequencing data: a comparison of mutation callers. Genome Med 5:91
8. Xu H, DiCarlo J, Satya RV et al (2014) Comparison of somatic mutation calling methods in amplicon and whole exome sequence data. BMC Genomics 15:244
9. Gerlinger M, Rowan AJ, Horswell S et al (2012) Intratumor heterogeneity and branched evolution revealed by multiregion sequencing. N Engl J Med 366:883–892
10. Jacoby MA, Duncavage EJ, Walter MJ (2015) Implications of tumor clonal heterogeneity in the era of next-generation sequencing. Trends Cancer 1:231–241
11. Pleasance ED, Cheetham RK, Stephens PJ et al (2010) A comprehensive catalogue of somatic mutations from a human cancer genome. Nature 463:191–196
12. Alexandrov LB, Nik-Zainal S, Wedge DC et al (2013) Signatures of mutational processes in human cancer. Nature 500:415–421
13. Roth A, Ding J, Morin R et al (2012) JointSNVMix: a probabilistic model for accurate detection of somatic mutations in normal/tumour paired next-generation sequencing data. Bioinformatics 28:907–913
14. Saunders CT, Wong WS, Swamy S et al (2012) Strelka: accurate somatic small-variant calling from sequenced tumor-normal sample pairs. Bioinformatics 28:1811–1817
15. Cibulskis K, Lawrence MS, Carter SL et al (2013) Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. Nat Biotechnol 31:213–219
16. The Broad Institute (2018.) https://software.broadinstitute.org/gatk/. Accessed 08 Jan 2018
17. Cingolani P (2017) SnpEff: genomic variant annotations and functional effect prediction toolbox. http://snpeff.sourceforge.net/. Accessed 08 Jan 2018
18. Koboldt DC, Zhang Q, Larson DE et al (2012) VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res 22:568–576
19. Cock PJ, Fields CJ, Goto N et al (2010) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res 38:1767–1771
20. Poplin R, Ruano-Rubio V, DePristo MA, et al (2017) Scaling accurate genetic variant discovery to tens of thousands of samples. https://doi.org/10.1101/201178. Accessed 08 Jan 2018
21. Garrison E and Marth G (2012) Haplotype-based variant detection from short-read sequencing. arXiv:1207.3907v2.: https://arxiv.org/abs/1207.3907. Accessed 08 Jan 2018
22. Babraham Bioinformatics (2017) .FastQC: a quality control tool for high throughput sequence data. https://www.bioinformatics.babraham.ac.uk/projects/fastqc/. Accessed 8 Jan 2018
23. Ewels P, Magnusson M, Lundin S et al (2016) MultiQC: summarize analysis results for multiple tools and samples in a single report. Bioinformatics 32:3047–3048
24. Bolger AM, Lohse M, Usadel B (2014) Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics 30:2114–2120
25. Li H, Durbin R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25:1754–1760
26. Benjamin D (2017) Pair HMM probabilistic realignment in HaplotypeCaller and Mutect. https://github.com/broadinstitute/gatk/blob/master/docs/pair_hmm.pdf. Accessed 08 Jan 2018
27. Benjamin D, Sato T (2018) Mathematical notes on mutect. https://github.com/broadinstitute/gatk/blob/master/docs/mutect/mutect.pdf. Accessed 08 Jan 2018
28. Benjamin D (2017) Local assembly in HaplotypeCaller and Mutect. https://github.com/broadinstitute/gatk/blob/master/docs/local_assembly.pdf. Accessed 08 Jan 2018
29. Sherry ST, Ward MH, Kholodov M et al (2001) dbSNP: the NCBI database of genetic variation. Nucleic Acids Res 29:308–311
30. Consortium GP, Auton A, Brooks LD, et al (2015) A global reference for human genetic variation. Nature 526:68-74
31. Lek M, Karczewski KJ, Minikel EV et al (2016) Analysis of protein-coding genetic variation in 60,706 humans. Nature 536:285–291
32. GnomAD. Browser beta, genome aggregation database (2017.) http://gnomad.broadinstitute.org/. Accessed 10 Jan 2018
33. Danecek P, Auton A, Abecasis G et al (2011) The variant call format and VCFtools. Bioinformatics 27:2156–2158
34. Cingolani P, Platts A, Wang le L, et al (2012) A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila

melanogaster strain w1118; iso-2; iso-3. Fly (Austin) 6:80-92
35. Cingolani P, Patel VM, Coon M et al (2012) Using Drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, SnpSift. Front Genet 3:35
36. McLaren W, Gil L, Hunt SE et al (2016) The Ensembl variant effect predictor. Genome Biol 17:122
37. Wang K, Li M, Hakonarson H (2010) ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Res 38:e164
38. Golden Helix SNP & Variation Suite™ (2017) Golden Helix, Inc., Bozeman, MT. http://www.goldenhelix.com/. Accessed 15 Jan 2018
39. Eilbeck K, Lewis SE, Mungall CJ et al (2005) The Sequence Ontology: a tool for the unification of genome annotations. Genome Biol 6:R44
40. Liu X, Jian X, Boerwinkle E (2011) dbNSFP: a lightweight database of human nonsynonymous SNPs and their functional predictions. Hum Mutat 32:894–899
41. Liu X, Wu C, Li C et al (2016) dbNSFP v3.0: a one-stop database of functional predictions and annotations for human nonsynonymous and Splice-Site SNVs. Hum Mutat 37:235–241
42. Landrum MJ, Lee JM, Benson M et al (2016) ClinVar: public archive of interpretations of clinically relevant variants. Nucleic Acids Res 44:D862–D868
43. Gates C and Bene J (2016) .Jacquard: a suite of command-line tools to expedite analysis of exome variant data from multiple patients and multiple variant callers. https://github.com/umich-brcf-bioinf/Jacquard. Accessed 08 Jan 2018
44. Kim SY, Jacob L, Speed TP (2014) Combining calls from multiple somatic mutation-callers. BMC Bioinformatics 15:154
45. Fang LT, Afshar PT, Chhibber A et al (2015) An ensemble approach to accurately detect somatic mutations using SomaticSeq. Genome Biol 16:197
46. Callari M, Sammut SJ, De Mattos-Arruda L et al (2017) Intersect-then-combine approach: improving the performance of somatic variant calling in whole exome sequencing data using multiple aligners and callers. Genome Med 9:35