Sequencing Solutions Technical Note

August 2015

How To…

Evaluate NimbleGen SeqCap EZ Target Enrichment Data

Applications
Targeted Sequencing

Products
SeqCap EZ Exome (all versions)
SeqCap EZ Developer
SeqCap EZ Choice
SeqCap EZ Designs

1. Overview

Analysis of NimbleGen SeqCap EZ target enrichment experimental data sequenced on an Illumina sequencing system is most frequently performed using a variety of publicly available, open-source analysis tools.

The usage examples described here have been used effectively in our hands. *Please note that publicly available, open-source software tools may change and that such change is not under the control of Roche. Therefore Roche does not warrant and cannot be held liable for the results obtained when using the third party tools described herein. Roche does not provide direct analysis support or service for these or any other third party tools. Please refer to the authors of each tool for support and documentation.*

The typical variant calling analysis workflow consists of sequencing read quality assessment, read filtering, mapping against the reference genome, duplicate removal, coverage statistic assessment, variant calling, and variant filtering. At most of these steps, a variety of tools is available to do the job. This document shows how to use a selection of the available tools to perform SeqCap EZ data analysis, but other analysis workflows can also be used.

This document will enable readers with bioinformatics experience to understand the basic analysis workflow in use at Roche NimbleGen to assess capture performance. The reader should carefully consider additional options when deciding the most appropriate workflow for their research.

Changes in this Version

Major changes since the last version of this document include:

- Software version and usage updates for almost all tools used
- Added a flowchart of the main workflow
- Added new tools SAMtools fixmate, Picard BedToIntervalList, and GATK GenotypeConcordance
- Added the Description of Metrics table
- Added definition for empirical target
- Fixed the on-target read count command line (previously generated a BED file, not a count)
- Removed sections on working with two-track BED files

2. Solution

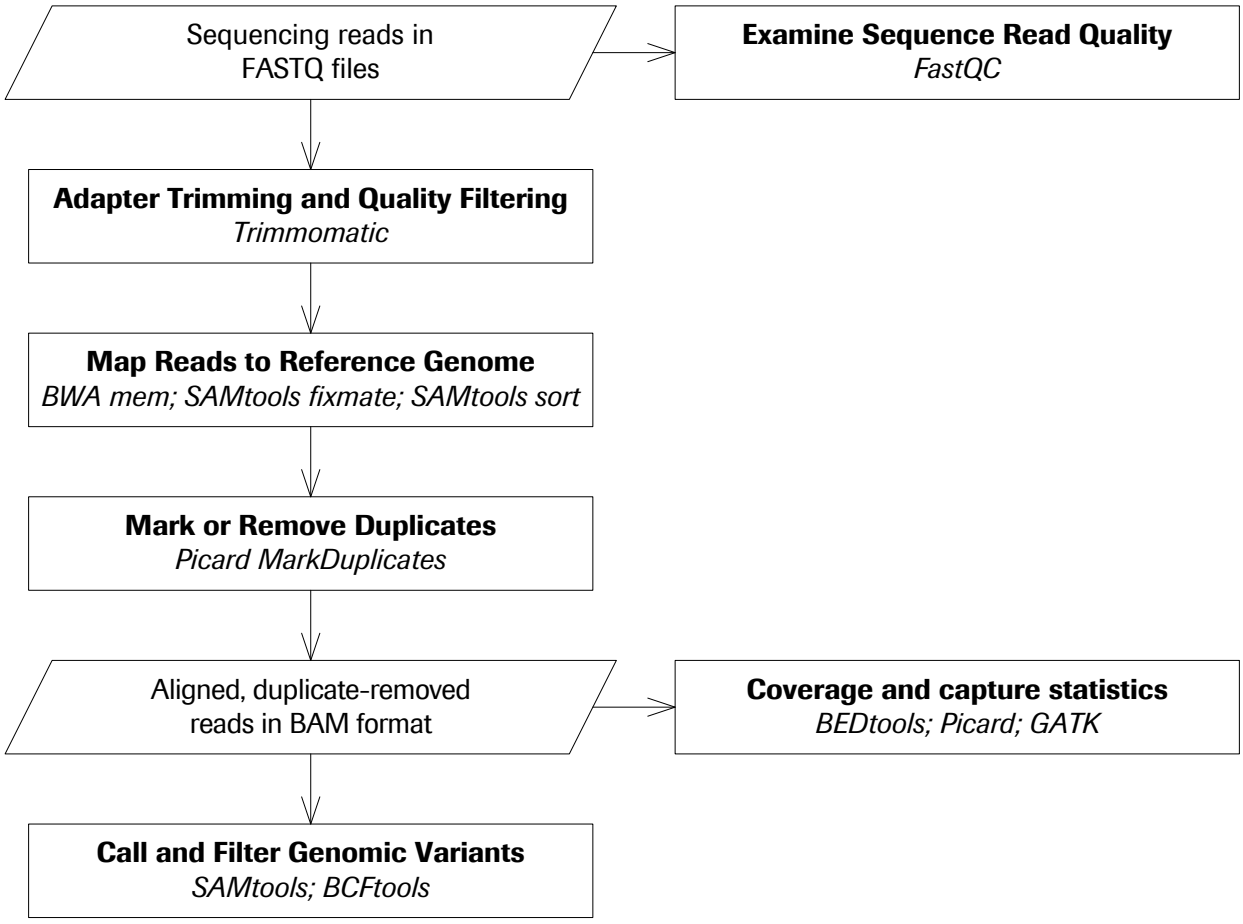

Figure 1. Schematic of basic SeqCap EZ analysis workflow.

Free and open source third-party tools are available for converting raw sequencing data into appropriate file formats, mapping reads to a reference sequence, evaluating sequencing quality, and analyzing variant calls. This technical note describes a number of steps and mini-workflows that use such third-party tools, which can be combined together into a variety of data analysis workflows.

Ideally, you should develop a workflow appropriate for your experimental data using benchmark/control samples, such as HapMap samples obtained from Coriell. Known variants for HapMap samples may be downloaded from the HapMap project (*http://hapmap.ncbi.nlm.nih.gov*), the 1000 Genomes Project (*http://www.1000genomes.org*), or in special collections such as the GATK resource bundle (*http://www.broadinstitute.org/gatk/download*) and Genome in a Bottle (*https://sites.stanford.edu/abms/giab*).

Note that where the text 'SAMPLE' appears throughout examples shown here, you should replace it with a unique sample name. Similarly, replace 'DESIGN' with the name of the SeqCap EZ target enrichment design that matches the design files supplied by Roche NimbleGen.

Replace '/path/to/…' in the examples with a valid path on your system. The current directory is assumed to be the location of all input files, and will also be the location of output files and report files.

Type the entire command shown for each step on a single line, despite the way it appears on the printed page. There should be no spaces within a file path, but there must be spaces before and after each option. *Tip*: if supported on your system, use the Tab key to auto-complete paths and file names while typing.

Due to quirks in most if not all PDF viewers, the underscores in command line examples do not display properly at all zoom percentages. One way to confirm whether or not underscores are present is to print the page. Alternatively, try temporarily switching to a very high zoom percentage (*e.g.* 400%).

Examples included in this document show how to perform the analysis with paired end Illumina sequencing reads. Many of the tools work with single end reads if paired end reads are not available. Note that using single end reads will artificially increase duplicate rate due to decreased ability to resolve a unique fragment from the library.

Tools Overview

Package (version)	Tool	Function as used in this document
BCFtools (1.2)	`View`	Convert between VCF and BCF formats.
	`Call`	Variant calling and filtering.
	Filter	Filter variant calls.
BEDtools2 (2.24.0)	`Intersect`	Calculate on-target read rate by intersecting a list of regions in BED format against mapped reads in BAM format.
	`Sort`	Sort BED formatted regions.
	`Merge`	Merge overlapping regions within a BED file.
	`Genomecov`	Calculate sum total size of regions in a BED file.
	`Slop`	Pad (extend) length of target regions.
BWA (0.7.12-r1039)	`Index`	Generate an indexed genome from FASTA sequence.
	`Mem`	Map sequencing reads to an indexed genome.
FastQC (0.11.3)	`Fastqc`	Assess sequencing read quality (per-base quality plot).
GATK (public tools) (3.4)	`DepthOfCoverage`	Calculate mean, median, and specific sequencing coverage depths.
	`GenotypeConcordance`	Compare SNP calls against a set of known SNPs for the sample used.
IGV	`Igv`	Genomic viewer for BAM and BED files (not explicitly used in the examples shown in this document). See *References* for a link to additional information.
Picard (1.134)	`BedToIntervalList`	Convert BED file to Picard Interval List format.
	`CreateSequenceDictionary`	Generate a sequence dictionary (.dict) for the reference genome.
	`CollectInsertSizeMetrics`	Estimate insert size mean and standard deviation and plot an insert size distribution.
	`CalculateHsMetrics`	Assess performance of a target enrichment experiment based on mapped reads.
	`MarkDuplicates`	Remove or mark duplicate reads, and count the number of paired, unpaired, and optical duplicates.
	`CollectAlignmentSummaryMetrics`	Report mapping metrics for a BAM file.
	`VcfFormatConverter`	Reformat a version 4.2 VCF file to appear as version 4.1.
SAMtools (1.2)	`sort`	Sort a BAM file.
	`faidx`	Generate a FASTA index of the reference genome.
	`view`	Convert between SAM and BAM file formats.
	`mpileup`	Call variants on a BAM file.
	`fixmate`	Clean up paired read information.
seqtk (1.0-r31)	`sample`	Randomly subsample FASTQ file(s).
Trimmomatic (0.33)	`trimmomatic`	Trim raw reads for quality.

Table 1: Third-party data analysis tools used in this technical note. The examples described in this document were tested using the software versions listed in parentheses. See links in *References* for installation instructions and explanations of command options. These tools were tested on a Redhat Linux system.

Index a Reference Genome

Most Next-Generation Sequencing (NGS) mapping algorithms require an indexed genome be created before mapping. Although algorithms work in different ways, most use the Burrows-Wheeler algorithm for mapping millions of relatively short reads against the reference genome. A genomic index is used to very quickly to find the mapping location. Genomic indexing is required only once per genome version. The genomic index files that are created can then be used for all subsequent mapping jobs against that genome assembly version

Index the FASTA formatted genome sequence with chromosomes in 'karyotypic' sort order, *i.e.* chr1, chr2, ..., chr10, chr11, … chrX, chrY, chrM, chr1_random, *etc.* In these examples, reference genome files are referred to as 'ref.fa', which should be replaced by the actual file name (*e.g.* 'hg19.fa').

Package⇨Tool(s) Used	BWA⇨`index` SAMtools⇨`faidx` Picard⇨`CreateSequenceDictionary`
Input(s)	ref.fa
Output(s)	ref.fa {indexed} ref.fa = unmodified reference genome ref.fa.amb, ref.fa.ann, ref.fa.bwt, ref.fa.pac, ref.fa.sa = reference genome index files ref.fa.fai = FASTA index ref.dict = reference sequence dictionary

Generate Reference Genome Index

```
/path/to/bwa index -a bwtsw /path/to/ref.fa
```

Generate FASTA Index

```
/path/to/samtools faidx /path/to/ref.fa
```

Generate Sequence Dictionary

```
java -Xmx4g -Xms4g -jar /path/to/picard.jar CreateSequenceDictionary
REFERENCE=/path/to/ref.fa OUTPUT=ref.dict
```

The requirement for use of an indexed reference genome in a subsequent step is designated by 'ref.fa {indexed}' in the Input(s) section. An indexed reference genome consists of the genome FASTA file and all index files present in the same directory.

Decompress a FASTQ File

If the FASTQ files have been compressed (with a .gz extension), some tools require them to be decompressed before use.

Package⇨Tool(s) Used	`gunzip`
Input(s)	SAMPLE_R1.fastq.gz SAMPLE_R2.fastq.gz
Output(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq

```
gunzip -c SAMPLE_R1.fastq.gz > SAMPLE_R1.fastq

gunzip -c SAMPLE_R2.fastq.gz > SAMPLE_R2.fastq
```

Select a Subsample of Reads from a FASTQ File

Random subsampling is useful for normalizing the number of reads per set when doing comparisons. With paired end reads, it is important that the two files use the same values for the seed (`-s`) and number of reads. The seqtk application can optionally sample from gzipped FASTQ files, but will write the sampled reads to uncompressed FASTQ files.

Package⇨Tool(s) Used	seqtk⇨`sample`
Input(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq
Output(s)	SAMPLE_subset_R1.fastq SAMPLE_subset_R2.fastq

```
/path/to/seqtk sample -s 10000 SAMPLE_R1.fastq 30000000 > SAMPLE_subset_R1.fastq

/path/to/seqtk sample -s 10000 SAMPLE_R2.fastq 30000000 > SAMPLE_subset_R2.fastq
```

The commands above will randomly subsample 30 million matched read pairs from the paired end FASTQ files for a total of 60 million reads. Supplying the same random seed value (`-s`) ensures that the FASTQ records will remain in synchronized sort order and can be used for mapping, *etc*. Note that seqtk requires an amount of RAM proportional to the number of reads being subsampled. As you increase the size of the subsampled read set, more RAM is needed.

Examine Sequence Read Quality

Before spending time evaluating mapping statistics, use `fastqc` to run a series of tests on raw reads and generate a per-base sequence quality plot and report. The `fastqc` tool can work on both compressed and uncompressed FASTQ files.

Package⇨Tool(s) Used	FastQC⇨`fastqc`
Input(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq
Output(s)	SAMPLE_R1_fastqc.zip SAMPLE_R2_fastqc.zip

```
/path/to/fastqc --nogroup SAMPLE_R1.fastq SAMPLE_R2.fastq
```

A .zip file is created for each SAMPLE input file. An HTML report named fastqc_report.html is created that is viewable in an internet browser. The authors of FastQC have posted the following examples of the QC report for a good and a bad sequencing run:
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/good_sequence_short_fastqc.html
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/bad_sequence_fastqc.html

Adapter trimming and quality filtering

Before mapping reads to the reference genome, it is advisable to trim off any sequencing adapters found on the reads, and to filter or trim the reads for sequence quality. The Trimmomatic application can do both of those steps. The 'adapters.fa' file is a FASTA file containing sequencing adapters. See the Trimmomatic download 'adapters' subdirectory for examples. The Trimmomatic-provided file 'TruSeq3-PE-2.fa' is recommended. For custom adapters you may need to make your own FASTA file. If using subsampled reads, the subsampled FASTQ files should be used as input here.

Package⇨Tool(s) Used	Trimmomatic
Input(s)	adapters.fa SAMPLE_R1.fastq (shown below) or SAMPLE_R1_subset.fastq if subsampling SAMPLE_R2.fastq (shown below) or SAMPLE_R2_subset.fastq if subsampling
Output(s)	SAMPLE_R1_trimmed.fastq SAMPLE_R2_trimmed.fastq SAMPLE_R1_unpaired.fastq SAMPLE_R2_unpaired.fastq trimmomatic.log

Trim Reads (optional, but recommended)

```
java -Xms4g -Xmx4g -jar /path/to/trimmomatic.jar PE -threads NumProcessors -phred33
SAMPLE_R1.fastq SAMPLE_R2.fastq SAMPLE_R1_trimmed.fastq SAMPLE_R1_unpaired.fastq
SAMPLE_R2_trimmed.fastq SAMPLE_R2_unpaired.fastq
ILLUMINACLIP:/path/to/adapters.fa:2:30:10 TRAILING:20 SLIDINGWINDOW:5:20 MINLEN:50
&> trimmomatic.log
```

The Trimmomatic application will produce four files. The SAMPLE_R1_trimmed.fq and SAMPLE_R2_trimmed.fq contain the reads that are still paired after adapter trimming and quality filtering. The SAMPLE_R1_unpaired.fq and SAMPLE_R2_unpaired.fq contain singleton reads, where the other mate of the pair was discarded because of poor quality or because the remaining read was shorter than the minimum length of 50 bp. The unpaired reads do not have to be discarded but for most alignment applications they will have to be mapped separately from the paired reads, which can complicate downstream steps in a workflow. If you want to increase the percentage of passing reads, consult the Trimmomatic user guide and adjust the Trimmomatic parameters (especially `MINLEN`, which is the required minimum length after trimming).

Map Reads to Reference Genome

Reads are mapped to the indexed reference genome using BWA. SAMtools `fixmate` ensures consistent information appears for both reads in a pair. SAMtools sort is then used to order the output file according to genomic sort order.

Package⇨Tool(s) Used	BWA⇨`mem` SAMtools⇨`view` SAMtools⇨`fixmate` SAMtools⇨`sort`
Input(s)	ref.fa {indexed} SAMPLE_R1_trimmed.fastq SAMPLE_R2_trimmed.fastq
Output(s)	SAMPLE_sorted.bam

Map Reads

```
/path/to/bwa mem -R "@RG\tID:1\tPL:illumina\tLB:SAMPLE\tSM:SAMPLE" /path/to/ref.fa -
t NumProcessors -M SAMPLE_R1_trimmed.fq SAMPLE_R2_trimmed.fq | /path/to/samtools
view -Sb - > SAMPLE_bwa.bam
```

Clean up paired read information

```
/path/to/samtools fixmate -O bam SAMPLE_bwa.bam SAMPLE_fixmate.bam
```

Sort BAM File

```
/path/to/samtools sort -@ NumProcessors -O bam -o SAMPLE_sorted.bam -T
/tmp/samtools_sort_tmp SAMPLE_fixmate.bam
```

In the 'Map Reads' step, the `-R` option defines the read group ('@RG'), which will appear in the BAM header. Within this string is the sample ID ('ID'), sequencing platform ('PL'), library name ('LB'), and sample name ('SM'). When a library name, ID and sample name do not separately exist, a SAMPLE descriptor may be used, as shown in the example above.

Basic Mapping Metrics (Picard)

Basic mapping metrics can be calculated using Picard `CollectAlignmentSummaryMetrics` with SAMPLE.bam and ref.fa {indexed} files as input.

Package⇨Tool(s) Used	Picard⇨`CollectAlignmentSummaryMetrics`
Input(s)	ref.fa {indexed} SAMPLE_sorted.bam
Output(s)	SAMPLE_picard_alignment_metrics.txt

```
java -Xmx4g -Xms4g -jar /path/to/picard.jar CollectAlignmentSummaryMetrics
METRIC_ACCUMULATION_LEVEL=ALL_READS INPUT=SAMPLE_sorted.bam
OUTPUT=SAMPLE_picard_alignment_metrics.txt REFERENCE_SEQUENCE=/path/to/ref.fa
VALIDATION_STRINGENCY=LENIENT
```

See *https://broadinstitute.github.io/picard/picard-metric-definitions.html* for a description of the output metrics.

Mark or Remove Duplicates

After mapping, the Picard MarkDuplicates command is used to remove or mark PCR duplicates. This is done to avoid allele amplification bias in variant calling.

Package⇨Tool(s) Used	Picard⇨`MarkDuplicates`
Input(s)	SAMPLE_sorted.bam
Output(s)	SAMPLE_sorted_rmdups.bam SAMPLE_sorted_rmdups.bai SAMPLE_picard_markduplicates_metrics.txt

Mark Duplicates

```
java -Xmx4g -Xms4g -jar /path/to/picard.jar MarkDuplicates
VALIDATION_STRINGENCY=LENIENT INPUT=SAMPLE_sorted.bam
OUTPUT=SAMPLE_sorted_rmdups.bam
METRICS_FILE=SAMPLE_picard_markduplicates_metrics.txt REMOVE_DUPLICATES=true
ASSUME_SORTED=true CREATE_INDEX=true
```

In the 'Mark Duplicates' step, `REMOVE_DUPLICATES=true` means that duplicates are removed rather than marked. If you are certain that subsequent tools will appropriately understand the duplicate flag, you may alternatively use `REMOVE_DUPLICATES=false` (a duplicate flag is added for duplicate reads) to keep a record in the BAM file of which reads were considered duplicates.

View the file 'SAMPLE_picard_markduplicates_metrics.txt' for counts of paired, unpaired, and duplicate reads. See *https://broadinstitute.github.io/picard/picard-metric-definitions.html* for a description of the output metrics. Note that 'optical duplicates' are also reported, based on sequence similarity and sequencing cluster distance. Optical duplicates are not separately included in the total duplicate rate but are counted within the paired and unpaired duplicates.

The sorted and duplicate-marked SAMPLE.bam file is an input for many of the examples below. A requirement for use of an indexed BAM file in a subsequent step is designated by 'SAMPLE.bam {indexed}' in the Input(s) section. The BAM file is indexed if the SAMPLE.bam.bai index file is present in the same directory.

Estimate Insert Size Distribution

The DNA that goes into sequence capture is generated by random fragmentation, and later size selected. It is normal to observe a range of fragment sizes, but if skewed too large or too small the on-target rate and/or percent of bases covered with at least one read can be adversely affected.

Package⇨Tool(s) Used	Picard⇨`CollectInsertSizeMetrics`
Input(s)	SAMPLE_sorted_rmdups.bam
Output(s)	SAMPLE_picard_insert_size_metrics.txt SAMPLE_picard_insert_size_plot.pdf

```
java -Xmx4g -jar /path/to/picard.jar CollectInsertSizeMetrics
VALIDATION_STRINGENCY=LENIENT HISTOGRAM_FILE=SAMPLE_picard_insert_size_plot.pdf
INPUT=SAMPLE_sorted_rmdups.bam OUTPUT=SAMPLE_picard_insert_size_metrics.txt
```

See *https://broadinstitute.github.io/picard/picard-metric-definitions.html* for a description of output metrics included in SAMPLE_picard_insert_size_metrics.txt, which can also be used to plot the insert size distributions across samples. As long as R is installed on your system, a PDF plot is also created and placed in SAMPLE_picard_insert_size_plot.pdf.

Add Padding to Capture Target Regions

Target-adjacent coverage is typical for hybridization-based target enrichment due to the capture of partially on-target DNA library fragments. To optionally assess the amount of off-target reads which are target adjacent, follow the steps below to first pad the targets before assessing the on-target rate, as described in *Count On-Target Reads*. When the BEDtools input parameter is given a dash character ('`-i -`'), BEDtools will then take input from STDIN instead of expecting an input filename, allowing commands to be connected with a pipe.

Package⇨Tool(s) Used	BEDtools⇨`sort` BEDtools⇨`slop` BEDtools⇨`merge`
Input(s)	DESIGN_capture_targets.bed chromosome_sizes.txt
Output(s)	DESIGN_capture_targets_padded.bed

```
/path/to/bedtools sort -i DESIGN_capture_targets.bed | /path/to/bedtools slop -i -
-b 100 -g chromosome_sizes.txt | bedtools merge -i - >
DESIGN_capture_targets_padded.bed
```

In the BEDtools `slop` command, the value supplied to the `-b` option indicates the number of target-adjacent bases to add on both sides of the target. In this example, 100 bp would be added to both sides of all targets. Although 100 bp is commonly used for this kind of padding, shorter or longer lengths may also be appropriate depending on expected library fragment sizes.

The chromosome_sizes.txt file should be in the format: ChrName<tab>ChrSize, *e.g.* 'chr1 249250621' and have an entry for each chromosome present in the input BED file.

Please note: all remaining steps in this document are written to use non-padded targets. To use padded targets, replace the suggested design file for each step with your padded design file.

Determine Sum Total Size of Regions in a BED File

Use BEDtools `genomecov` to calculate the size in bases of a design BED file. The chromosome_sizes.txt file should be as described in *Add Padding to Capture Target*.

Package⇨Tool(s) Used	BEDtools⇨`genomecov`
	`grep`
	`cut`

Input(s)	chromosome_sizes.txt DESIGN_capture_targets.bed (example shown below) or DESIGN_primary_targets.bed
Output(s)	{size}

```
/path/to/bedtools genomecov -i DESIGN_capture_targets.bed -g chromosome_sizes.txt
-max 1 | grep -P "genome\t1" | cut -f 3
```

Count On-Target Reads

Use BEDtools `intersect` to calculate the number of reads which overlap a target BED file by at least 1 bp. Calculation of on-target reads is one measure of the success of a SeqCap EZ target enrichment experiment, though optimal on-target is design-specific. The on-target metric is affected by library insert size, hybridization and wash stringency, design size, and laboratory protocol.

Package⇨Tool(s) Used	BEDtools⇨`intersect` `wc`
Input(s)	SAMPLE_sorted_rmdups.bam DESIGN_primary_targets.bed or DESIGN_capture_targets.bed or DESIGN_capture_targets_padded.bed
Output(s)	on_target_reads.txt {count of on-target reads}

```
/path/to/bedtools intersect -bed -u -abam SAMPLE_sorted_rmdups.bam -b
DESIGN_primary_targets.bed | wc -l > on_target_reads.txt

or

/path/to/bedtools intersect -bed -u -abam SAMPLE_sorted_rmdups.bam  -b
DESIGN_capture_targets_padded.bed | wc -l > on_target_reads.txt
```

The command will output the number of on-target reads, counting as on-target any read that overlaps the target by at least one base. Divide this number by the total number of mapped, non-duplicate reads to get the percentage of on-target reads ('on-target rate'). See *Basic Mapping Metrics (Picard)* for reporting of the total number of mapped, non-duplicate reads.

Calculate Depth of Coverage

The GATK Framework includes a walker (utility) which is useful in calculating depth of coverage metrics over enrichment targets, including mean, median, and percent of target bases covered at requested depths. In the example below, percent of bases covered by at least one read (≥1X), at least 10 reads (≥10X), and at least 20 reads (≥20X) is calculated. To calculate percent of bases covered at other depths include any number of additional '`-ct`' options listing the desired depth.

Package⇨Tool(s) Used	GATK⇨`DepthOfCoverage`
Input(s)	ref.fa {indexed} SAMPLE_sorted_rmdups.bam {indexed}
Output(s)	SAMPLE_gatk_primary_target_coverage.sample_summary {there are other output files not listed here}

```
java -Xmx4g -Xms4g -jar /path/to/GATKFramework/GenomeAnalysisTK.jar -T
DepthOfCoverage -R /path/to/ref.fa -I SAMPLE_sorted_rmdups.bam -o
SAMPLE_gatk_primary_target_coverage -L DESIGN_primary_targets.bed -ct 1 -ct 10 -ct
20
```

The sample summary output file reports the mean and median depth of coverage over the given regions of interest, as well as the percent of bases in the given regions covered at or above a given depth. For more information, consult the documentation at:
https://www.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_gatk_tools_walkers_coverage_DepthOfCoverage.php

Call and Filter Genomic Variants

Once reads are mapped, calling variants against the reference genome is most often the goal. Due to errors which may be accumulated during amplification steps and during sequencing, it's recommended to apply filters to called variants. This includes requiring a minimum depth of coverage to call a variant and may also include a mapping quality filter. Omitting or lowering the stringency of filters will result in a larger number of false variant calls. Increasing filter stringency will decrease the overall number of variant calls but should result in higher confidence calls.

Use the SAMtools `mpileup` and BCFtools `call` commands to call variants (SNPs and small indels), and use the BCFtools `filter` command to filter the variants by read depth.

Package⇨Tool(s) Used	SAMtools⇨`mpileup` BCFtools⇨`call` BCFtools⇨`view` BCFtools⇨`filter`
Input(s)	ref.fa {indexed} SAMPLE_sorted_rmdups.bam {indexed}
Output(s)	SAMPLE_filtered_variants.vcf

Call Genomic Variants

```
/path/to/samtools mpileup -Bugf /path/to/ref.fa SAMPLE_sorted_rmdups.bam |
/path/to/bcftools call -vm -O u -o SAMPLE_samtools_var.raw.bcf
```

Filter Raw Variants

```
/path/to/bcftools filter -i 'MQ>=30 && DP>=12 && DP<=10000'
SAMPLE_samtools_var.raw.bcf > SAMPLE_filtered_variants.vcf
```

The BCFtools `filter` command is being used to filter variant calls to remove low confidence calls. Mapping quality is filtered by 'MQ>=30' in the example. Reads which don't map uniquely in the genome automatically receive a mapping quality score of 0, therefore this filter will remove from variant calling any non-unique regions which may have been sequenced. As the mapping quality is increased, the number of variants called decreases. Also in the example, 'DP>=12' and 'DP<=10000' are filtering on minimum and maximum read depth, respectively. As you increase the minimum read depth you will start to lose true variant calls with low coverage, but variants with a depth of fewer than five reads are generally unreliable due to sequencing error.

BCFtools creates a VCF file according to the VCF file format specification version 4.2. Older versions of GATK appear to work only with v4.1 VCF files. If necessary, use Picard `VcfFormatConverter` to reformat the v4.2 VCF file to appear as v4.1 (command not shown).

Additional downstream variant analysis is not covered in this document, but may consist of comparison of SNP calls against dbSNP, variant classification, and variant effect analysis.

Create Picard Interval Lists

Picard interval lists are genomic interval description files required by the Picard `CalculateHsMetrics` utility that contain a SAM-like header describing the reference genome and a set of coordinates with strand and name for each interval. The Picard 'target interval' is equivalent to the 'primary target', and the Picard 'bait interval' is equivalent to the 'capture target'.

Use the Picard `BedToIntervalList` command to create Picard Interval List files from target BED files.

Package⇨Tool(s) Used	Picard⇨`BedToIntervalList`
Input(s)	DESIGN_primary_targets.bed DESIGN_capture_targets.bed ref.dict {one of the files in the indexed genome file set}
Output(s)	DESIGN_target_intervals.txt DESIGN_bait_intervals.txt

Create a Picard Target Interval List

```
java -Xmx4g -jar /path/to/picard.jar BedToIntervalList
INPUT=DESIGN_primary_targets.bed SEQUENCE_DICTIONARY=ref.dict
OUTPUT=DESIGN_target_intervals.txt
```

Create a Picard Bait Interval List

```
java -Xmx4g -jar /path/to/picard.jar BedToIntervalList
INPUT=DESIGN_capture_targets.bed SEQUENCE_DICTIONARY=ref.dict
OUTPUT=DESIGN_bait_intervals.txt
```

The DESIGN_target_intervals.txt and DESIGN_bait_intervals.txt files are used by the Picard `CalculateHsMetrics` command.

Hybrid Selection (HS) Analysis Metrics

The `CalculateHsMetrics` command calculates a number of metrics assessing the quality of target enrichment reads, including fold 80 base penalty which is a metric of sequencing depth uniformity over the targets.

Package⇨Tool(s) Used	Picard⇨`CalculateHsMetrics`
Input(s)	ref.fa {indexed} SAMPLE_sorted_rmdups.bam {indexed} DESIGN_target_intervals.txt DESIGN_bait_intervals.txt
Output(s)	SAMPLE_picard_hs_metrics.txt

```
java -Xmx4g -Xms4g -jar /path/to/picard.jar CalculateHsMetrics
BAIT_INTERVALS=DESIGN_bait_intervals.txt
TARGET_INTERVALS=DESIGN_target_intervals.txt  INPUT=SAMPLE_sorted_rmdups.bam
OUTPUT=SAMPLE_picard_hs_metrics.txt METRIC_ACCUMULATION_LEVEL=ALL_READS
REFERENCE_SEQUENCE=/path/to/ref.fa VALIDATION_STRINGENCY=LENIENT TMP_DIR=.
```

Supplying the same Picard interval file to both 'TARGET_INTERVALS' and 'BAIT_INTERVALS' parameters of Picard `CalculateHsMetrics` can change the outcome, depending on how different the primary target (target interval) and capture target (bait interval) regions are from each other. See *https://broadinstitute.github.io/picard/picard-metric-definitions.html* for a description of output metrics.

Compare to Known SNPs

Use GATK `GenotypeConcordance` to compare variant calls to known variants for the sample. This is most commonly done to compare variant calls against know variants for Coriell HapMap samples such as NA12878 (female CEU/CEPH) or NA12891 (male, CEU/CEPH).

The gold standard VCF file should consist of high confidence known variants for the sample being examined. It's important to note that some sources for known variants are more trustworthy than others. The best gold standard will be built using two or more sources and if possible, two or more technology platforms. This reduces source- and platform-specific systematic errors in your gold standard variant list. Known variants for HapMap samples can be downloaded from the HapMap project (*http://hapmap.ncbi.nlm.nih.gov*), the 1000 Genomes Project (*http://www.1000genomes.org*), or in special collections such as GATK's resource bundle (*http://www.broadinstitute.org/gatk/download*) and Genome in a Bottle (*https://sites.stanford.edu/abms/giab*).

Package⇨Tool(s) Used	GATK⇨`GenotypeConcordance`
Input(s)	ref.fa {indexed} SAMPLE_filtered_variants.vcf gold_standard.vcf DESIGN_primary_targets.bed
Output(s)	SAMPLE_genotype_concordance.txt

```
java -Xmx4g -jar /path/to/GATKFramework/GenomeAnalysisTK.jar -T GenotypeConcordance
-R /path/to/ref.fa -o SAMPLE_genotype_concordance.txt -eval:SAMPLE
SAMPLE_filtered_variants.vcf -comp:GOLD_STD gold_standard.vcf -moltenize -L
DESIGN_primary_targets.bed
```

The option `-eval` specifies an experimental variant call set (callset) to be evaluated, while the `-comp` callset is a previously existing set of sample-specific known variants used as a gold standard for comparison. The option `-moltenize` outputs tables in the 'moltenized' tab-delimited format, including counts of EVAL versus COMP genotype states. The option `-L` restricts comparison of the two callsets to the given list of regions (BED).

The output of `GenotypeConcordance` consists of five tables: *GenotypeConcordance_CompProportions*, *GenotypeConcordance_Counts*, *GenotypeConcordance_EvalProportions*, *GenotypeConcordance_Summary*, and *SiteConcordance_Summary*. Here we use only the output from the *GenotypeConcordance_Counts* table. See the GATK GenotypeConcordance documentation (*http://www.broadinstitute.org/gatk/index.php*) for details on all tables.

Output format and terms used in *GenotypeConcordance_Counts* table

The *GenotypeConcordance_Counts* table shows genotype counts for specified allele type categories. Following are descriptions of abbreviated terms used in this table. See Table 2 for an example of the *GenotypeConcordance_Counts* table.

NO_CALL: reported genotype in VCF is './.', indicating not enough data to make a call

HET: heterozygous

HOM_REF: homozygous reference

HOM_VAR: homozygous variant

UNAVAILABLE: variant is not called in this callset

MIXED: call made on one chromosome (in a diploid situation) but not the other, appears as './1' in VCF

Sample Name	EVAL	COMP	Genotype Count
NA12878	HET	HET	224
NA12878	HET	HOM_REF	0
NA12878	HET	HOM_VAR	0
NA12878	HET	MIXED	0
NA12878	HET	NO_CALL	0
NA12878	HET	UNAVAILABLE	398
NA12878	HOM_REF	HET	0
NA12878	HOM_REF	HOM_REF	0
NA12878	HOM_REF	HOM_VAR	0
NA12878	HOM_REF	MIXED	0
NA12878	HOM_REF	NO_CALL	0
NA12878	HOM_REF	UNAVAILABLE	0
NA12878	HOM_VAR	HET	2
NA12878	HOM_VAR	HOM_REF	0
NA12878	HOM_VAR	HOM_VAR	137
NA12878	HOM_VAR	MIXED	0
NA12878	HOM_VAR	NO_CALL	0
NA12878	HOM_VAR	UNAVAILABLE	171
NA12878	MIXED	HET	0
NA12878	MIXED	HOM_REF	0
NA12878	MIXED	HOM_VAR	0
NA12878	MIXED	MIXED	0
NA12878	MIXED	NO_CALL	0
NA12878	MIXED	UNAVAILABLE	0
NA12878	Mismatching_Alleles	Mismatching_Alleles	0
NA12878	NO_CALL	HET	0
NA12878	NO_CALL	HOM_REF	0
NA12878	NO_CALL	HOM_VAR	0
NA12878	NO_CALL	MIXED	0
NA12878	NO_CALL	NO_CALL	0
NA12878	NO_CALL	UNAVAILABLE	0
NA12878	UNAVAILABLE	HET	17
NA12878	UNAVAILABLE	HOM_REF	14
NA12878	UNAVAILABLE	HOM_VAR	3
NA12878	UNAVAILABLE	MIXED	0
NA12878	UNAVAILABLE	NO_CALL	0
NA12878	UNAVAILABLE	UNAVAILABLE	0

Table 2: Example of the moltenized GenotypeConcordance_Counts output table. The header row was added here to aid with explanation.

Formulas to Calculate Sensitivity and Specificity of SNP Calls

Below are formulas to calculate true positive calls, true negative calls, false positive calls, false negative calls, misclassified calls, sensitivity of detection, and specificity of classification based on the data provided in the *GenotypeConcordance_Counts* output table. Counts in the table are provided for each EVAL to COMP comparison (EVAL/COMP). For example, HET/HOM_REF refers to HET under the EVAL column and HOM_REF under the COMP column in the output table.

True Positive (TP):	HET/HET, HOM_VAR/HOM_VAR
True Negative (TN):	HOM_REF/HOM_REF
False Positive (FP):	HET/HOM_REF, HOM_VAR/HOM_REF
False Negative (FN):	HOM_REF/HET, HOM_REF/HOM_VAR, NO_CALL/HET, NO_CALL/HOM_VAR , UNAVAILABLE/HET, UNAVAILABLE/HOM_VAR
Misclassified (MC):	HET/HOM_VAR, HOM_VAR/HET

$$\text{Sensitivity of Detection} = \frac{\text{Het\_TP} + \text{Hom\_TP}}{\text{Het\_TP} + \text{Hom\_TP} + \text{FN}}$$

$$\text{Sensitivity of Classification} = \frac{\text{Het\_TP} + \text{Hom\_TP}}{\text{Het\_TP} + \text{Hom\_TP} + \text{MC}}$$

Description of Metrics

The tools used in this document generate output files which contain many metrics. There are some metrics which are frequently monitored to assess sequence capture performance. Table 3 describes many of these metrics, which tool(s) are used to generate the metrics, and additional mathematical or string parsing operations which may be necessary to obtain the final values.

Metric	Tool(s) used to obtain value (name of output file used)	Description	Metric name in tool's output file and/or calculation method
Total input reads	`Trimmomatic (trimmomatic.log)`	Number of reads prior to Trimmomatic processing for quality and adapter trimming	('Input Read Pairs:')*2
% input reads after filtering	`Trimmomatic (trimmomatic.log)`	Percent of total input reads remaining after Trimmomatic processing	(('Both Surviving:')*2)/(('Input Read Pairs:')*2)*100

% trimmed reads mapped	`Picard⇨CollectAlignmentSummaryMetrics (metrics file)`
	
	`Trimmomatic (trimmomatic.log)`
		Percentage of filtered reads which mapped anywhere in the genome	total_mapped_reads = Picard CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column
			
			total_filtered_read_pairs = Trimmomatic output 'Both Surviving'
			
			% trimmed reads mapped = (total_mapped_reads / total_filtered_read_pairs*2) * 100

Total duplicate rate	`Picard⇨MarkDuplicates (metrics file)`	Percentage of aligned reads identified as PCR duplicates, includes both paired and unpaired reads.	PERCENT_DUPLICATION*100

% reads on-target	`BEDtools⇨intersect (on_target_reads.txt)`
	
	`Picard⇨CollectAlignmentSummaryMetrics (metrics file)`
	
	`Picard⇨MarkDuplicates (metrics file)`
		Number of mapped, non-duplicate reads overlapping a target region by at least 1 base. No padding/buffering.	on_target_reads = value in on_target_reads.txt
			
			total_mapped_reads = Picard CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column
			
			unpaired_read_duplicates = Picard MarkDuplicates UNPAIRED_READ_DUPLICATES in metrics file
			
			read_pair_duplicates = Picard MarkDuplicates READ_PAIR_DUPLICATES in metrics file
			
			% mapped, non-duplicate reads on-target = (on_target_reads) / (total_mapped_reads - unpaired_read_duplicates - read_pair_duplicates*2) * 100

Metric	Tool(s) used to obtain value (name of output file used)	Description	Metric name in tool's output file and/or calculation method
Fold enrichment	Picard⇨CalculateHsMetrics (metrics file)	Fold enrichment of the capture target compared to the whole genome. In terms of the metrics in the Picard CalculateHsMetrics output file: (ON_BAIT_BASES / (ON_BAIT_BASES + NEAR_BAIT_BASES + OFF_BAIT_BASES)) / (BAIT_TERRITORY / GENOME_SIZE). In other words, the fraction of sequencing bases in the capture target divided by the fraction of total genomic bases in the capture target.	FOLD_ENRICHMENT
Fold 80 base penalty (uniformity)	Picard⇨CalculateHsMetrics (metrics file)	Fold additional sequencing required to bring 80% of bases to the mean. This is a measure of sequence depth uniformity, lower is better - typical exome is between 2.5-3.5. The best theoretical value is 1. Zero coverage regions are excluded. Another name for this metric is '1/nc80'. This metric is sensitive to the total number of reads.	FOLD_80_BASE_PENALTY
Mean insert size	Picard⇨CalculateInsertSizeMetrics (metrics file)	Mean estimated capture fragment insert size.	MEAN_INSERT_SIZE
Insert size std dev	Picard⇨CalculateInsertSizeMetrics (metrics file)	Standard deviation of the estimated capture fragment insert size.	STANDARD_DEVIATION
Mean target coverage	GATK⇨DepthOfCoverage (SAMPLE_gatk_primary_target_coverage.sample_summary)	Mean depth of coverage over the primary target	mean
Median target coverage	GATK⇨DepthOfCoverage (SAMPLE_gatk_primary_target_coverage.sample_summary)	Median depth of coverage over the primary target	granular_median

Metric	Tool(s) used to obtain value (name of output file used)	Description	Metric name in tool's output file and/or calculation method
% bases >= NX	GATK⇨DepthOfCoverage (SAMPLE_gatk_primary_target_coverage.sample_summary)	Percentage of primary target bases covered by N or more reads	%_bases_above_N
Number of filtered SNP calls	SAMtools; bcftools; (SAMPLE_filtered_variants.vcf)	Number of SNP calls after filtering	grep -v INDEL SAMPLE_filtered_variants.vcf | grep -P -v "^#"
Sensitivity of SNP detection	GATK⇨GenotypeConcordance (SAMPLE_genotype_concordance.txt)	What percentage of known variants within the target regions were detected? (HET_TP + HOM_TP)/(HET_TP + HOM_TP + HET_FN + HOM_FN)	From GenotypeConcordance_Counts table: (HET_HET+HOM_VAR_HOM_VAR)/(HET_HET+HOM_VAR_HOM_VAR+HOM_REF_HET+HOM_REF_HOM_VAR+UNAVAILABLE_HET+UNAVAILABLE_HOM_VAR+NO_CALL_HET+NO_CALL_HOM_VAR)*100
Specificity of SNP classification	GATK⇨GenotypeConcordance (SAMPLE_genotype_concordance.txt)	For those known variants detected, the percentage that had the correct zygosity (homozygous vs heterozygous). (HET_TP + HOM_TP)/(HET_TP + HOM_TP + HET_MISCLASSIFIED + HOM_MISCLASSIFIED)	From GenotypeConcordance_Counts table: (HET_HET+HOM_VAR_HOM_VAR)/(HET_HET+HOM_VAR_HOM_VAR+HET_HOM_VAR+HOM_VAR_HET)*100
Overall genotype concordance	GATK⇨GenotypeConcordance (SAMPLE_genotype_concordance.txt)	Concordance of called SNPs against the known genotype	From GenotypeConcordance_Summary table: Overall_Genotype_Concordance* 100

Table 3. Description of important metrics

3. REFERENCES

Roche NimbleGen is not responsible for the content of the following third-party websites.

- BEDtools2: *https://github.com/arq5x/bedtools2/releases*
- BWA: *https://github.com/lh3/bwa*
- FastQC: *http://www.bioinformatics.babraham.ac.uk/projects/fastqc/*
- GATK (Broad Institute): *http://www.broadinstitute.org/gatk/*
- GATK (Roche NimbleGen fork of GATK public tools, free to all): *https://github.com/NimbleGen/gatk*
- IGV: *http://www.broadinstitute.org/igv/*
- Picard: *http://broadinstitute.github.io/picard*
- SAMtools & BCFtools: *http://www.htslib.org/*
- seqtk: *https://github.com/lh3/seqtk*
- Trimmomatic: *http://www.usadellab.org/cms/?page=trimmomatic*

4. GLOSSARY

BAI file – BAM index file. For tools that require an indexed **BAM file,** the BAI file must be present in the same location as the BAM file.

Bait interval (Picard) – See **Capture target**.

BAM file – Compressed form of the **SAM file** format.

BCF file – Compressed form of the **VCF file** format.

BED file – File format for describing genomic regions/intervals. BED file start coordinates are 0-based.

bp – Abbreviation for base pair.

Capture target – as defined by Roche NimbleGen, these are the regions covered directly by one or more probes (the probe footprint). Older format NimbleGen BED files refer to these regions as the **Tiled regions**. These are equivalent to the **Bait intervals** referred to by Picard.

Empirical target – as defined by Roche NimbleGen, these are the regions which, when following the product User's Guide, Roche NimbleGen has seen reproducible sequencing coverage (coverage depth and percent of samples may be product specific). For capture products with an empirical target instead of a primary target, use the empirical target file wherever a primary target file is requested. See product's release notes for more information and guidance specific to the product.

FASTA file – A standard file format for describing nucleic acid sequences.

FASTQ file – A standard file format for describing sequencing reads that also includes base quality information.

Genomic index – A form of the reference genome sequence which enables faster comparisons during alignment.

Picard interval file – File format for describing genomic regions/intervals which also contains a header describing the reference genome. Picard interval file start coordinates are 1-based. See **Bait interval (Picard)** and **Target interval (Picard)**.

Primary target – as defined by Roche NimbleGen, these are the regions against which probes were designed. Older format NimbleGen BED files refer to these regions as simply **Target regions**. Typically these are identical to the original regions of interest with regions less than 100 bp expanded to 100 bp total to facilitate probe selection. These are equivalent to the **Target intervals** referred to by Picard. If an empirical targets file is provided instead of a primary targets file, use the empirical target file where a primary targets file is requested.

SAM file – Sequence Alignment / Map file; a community standard format for specifying sequencing read alignment to a reference genome.

Target interval (Picard) – see **Primary target**.

Target region – see **Primary target**.

Tiled region – see **Capture target**.

VCF file – Variant call format; a community standard format for specifying variant calls for one or more samples or populations against a reference genome.

Published by:
Roche Diagnostics GmbH
Sandhofer Straße 116
68305 Mannheim
Germany

07187009001 (2) 0815