Roche

Sequencing Solutions Technical Note

March 2014

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 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.

In addition to showing how to use raw FASTQ files to generate a BAM file and call variants, this document includes supporting workflows which show how to prepare and work with NimbleGen BED files and how to create Picard interval list files.

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

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. Please refer to the authors of each tool for support and documentation.*

2. SOLUTION

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 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*).

Note that where the text 'SAMPLE' appears throughout examples shown here, you should replace it with a unique sample name. Similarly, replace '/path/to/…' 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.

Tools Overview

Package (version)	Tool	Function as used in this document
BEDtools (2.17.0)	`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 (0.7.5a-r405)	`index`	Generate an indexed genome from FASTA sequence.
	`mem`	Map sequencing reads to an indexed genome.
FastQC (0.10.1)	`fastqc`	Assess sequencing read quality (per-base quality plot).
GATK Framework (2.7-2)	`DepthOfCoverage`	Calculate mean, median, and specific sequencing coverage depths.
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 (1.98)	`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 targeted enrichment reads.
	`MarkDuplicates`	Remove or mark duplicate reads, and detail the number of paired, unpaired, and optical duplicates.
	`CollectAlignmentSummaryMetrics`	Report mapping metrics for a BAM file.
	`SamToFastq`	Create FASTQ file(s) from a SAM or BAM file.
SAMtools (0.1.18)	`sort`	Sort a BAM file.
	`index`	Index a sorted BAM file.
	`faidx`	Generate a FASTA index of the reference genome.
	`flagstat`	Report mapping metrics for a BAM file.
	`view`	View or extract header or read data.
	`mpileup`	Call variants on a BAM file.
	BCFtools⇨`view`	Convert between VCF and BCF formats.
	vcfutils⇨`varFilter`	Filter raw variants.
seqtk (1.0-r31)	`sample`	Randomly subsample FASTQ file(s).
Trimmomatic (0.30)	`illuminaclip`	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 Linux system, but should also work on MacOS.

Index a Reference Genome

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/CreateSequenceDictionary.jar
REFERENCE=/path/to/ref.fa OUTPUT=ref.dict
```

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. 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
```

Generate FASTQ Files from a BAM File

If you don't have access to the original FASTQ files, it is possible to generate FASTQ files from a BAM file in order to remap the reads using a different pipeline.

Package⇨Tool(s) Used	Picard⇨`SamToFastq`
Input(s)	SAMPLE_file.bam
Output(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq

```
java -Xmx4g –Xms4g -jar /path/to/Picard/SamToFastq.jar VALIDATION_STRINGENCY=LENIENT
INPUT=SAMPLE_file.bam FASTQ=SAMPLE_R1.fastq SECOND_END_FASTQ=SAMPLE_R2.fastq
```

The generated FASTQ files will only be identical to the originals if the creator of the BAM file did not remove reads such as duplicates, low quality, *etc.*, or perform base quality recalibration.

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 10000000 > SAMPLE_subset_R1.fastq

/path/to/seqtk sample -s 10000 SAMPLE_R2.fastq 10000000 > SAMPLE_subset_R2.fastq
```

The commands above will randomly subsample 10 million read pairs from the paired end FASTQ files. 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 and decompressed directory are created for each SAMPLE input file. They contain an HTML report named fastqc_report.html 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/fastqc_report.html
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/bad_sequence_fastqc/fastqc_report.html

Generate a Sorted, Indexed BAM File

If necessary, follow the steps described in *Index a Reference* , *Decompress a FASTQ File*, and *Examine Sequence Read Quality Control* before using the steps described below. The 'adapters.fa' file is a FASTA file containing sequencing adapters (see the Trimmomatic download 'adapters' subdirectory for examples).

Package⇨Tool(s) Used	Trimmomatic⇨`illuminaclip` BWA⇨`mem` SAMtools⇨`view` SAMtools⇨`sort` Picard⇨`MarkDuplicates` SAMtools⇨`index`
Input(s)	adapters.fa ref.fa {indexed} SAMPLE_R1.fastq SAMPLE_R2.fastq
Output(s)	SAMPLE.bam {indexed} SAMPLE.bam = sorted and duplicate-marked BAM file SAMPLE.bam.bai = BAM index file SAMPLE_picard_markduplicates_metrics.txt

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.fq SAMPLE_R1_unpaired.fq
SAMPLE_R2_trimmed.fq SAMPLE_R2_unpaired.fq ILLUMINACLIP:/path/to/adapters.fa:2:30:10
LEADING:20 TRAILING:20 SLIDINGWINDOW:5:20 MINLEN:75
```

Map Reads
```
/path/to/bwa mem -R "@RG\tID:SAMPLE\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_unsorted.bam
```

Sort BAM File
```
/path/to/samtools sort SAMPLE_unsorted.bam SAMPLE_sorted
```

Mark Duplicates
```
java –Xmx4g –Xms4g -jar /path/to/Picard/MarkDuplicates.jar
VALIDATION_STRINGENCY=LENIENT INPUT=SAMPLE_sorted.bam OUTPUT=SAMPLE.bam
METRICS_FILE=SAMPLE_picard_markduplicates_metrics.txt REMOVE_DUPLICATES=false
ASSUME_SORTED=true
```

Index BAM File
```
/path/to/samtools index SAMPLE.bam
```

The parameters shown in the 'Trim Reads' step are best used with 2x100bp sequencing reads, where you should expect to see about 90% of reads pass these quality filters. If you want to increase the percentage of passing reads, adjust the Trimmomatic parameters (especially `MINLEN`, which is the required minimum length after trimming).

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.

In the 'Sort BAM File' step, do not add the '.bam' file extension to the output file name in the command line, because this extension is automatically added by the software.

In the 'Mark Duplicates' step, `REMOVE_DUPLICATES=true` may also be used. This may be desirable if it is unclear whether or not subsequent tools will appropriately understand the duplicate flag. View the file 'SAMPLE_picard_markduplicates_metrics.txt' for counts of paired, unpaired, and duplicate reads. See *http://picard.sourceforge.net/picard-metric-definitions.shtml* 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.

Basic Mapping Metrics (SAMtools)

Basic mapping metrics can be calculated using SAMtools, from the SAMPLE.bam file. If necessary, follow the steps described in *Generate a Sorted, Indexed BAM File* before using the step described below.

Package⇨Tool(s) Used	SAMtools⇨`flagstat`
Input(s)	SAMPLE.bam
Output(s)	SAMPLE_samtools_flagstat_metrics.txt

```
/path/to/samtools flagstat SAMPLE.bam > SAMPLE_samtools_flagstat_metrics.txt
```

The output file includes the number of total reads in the BAM file, number of mapped reads, number of reads that are paired and unpaired (singletons) in mapping, and number of duplicate reads (if marked instead of removed).

Basic Mapping Metrics (Picard)

Basic mapping metrics can be calculated using Picard, from SAMPLE.bam and ref.fa {indexed} files. If necessary, follow the steps described in *Index a Reference* and *Generate a Sorted, Indexed BAM File* before using the step described below.

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

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

See *http://picard.sourceforge.net/picard-metric-definitions.shtml* for a description of the output metrics.

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. If necessary, follow the steps described in *Generate a Sorted, Indexed BAM File* before using the step described below.

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

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

See *http://picard.sourceforge.net/picard-metric-definitions.shtml* 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.

Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files

The BED file format is a tab-delimited file that is flexible in the number of columns it contains (for additional detail, see *http://genome.ucsc.edu/FAQ/FAQformat.html#format1*). Only the first three columns (chromosome, start coordinate, end coordinate) are required. The BED files provided with SeqCap EZ designs include either a single two-track BED file or a pair of one-track BED files. Each track begins with a header line followed by coordinates for either (a) the 'primary target' (regions that capture probes were designed against, also called the 'target region') or (b) the 'capture target' (footprint of overlapping capture probes, also called the 'tiled region').

For use with analysis tools, two-track BED files are split into separate tracks, all track header lines are removed, region coordinates are sorted, and then overlapping/book-ended regions (pairs of regions where one region starts immediately after the end of another) are merged. It is good practice to check the first and last few lines of each extracted BED file to verify that the extraction was successful.

In the specific BED files used in the examples below, the primary target track (target region) starts with a header line, followed by 242,305 lines of coordinates. The capture target track (tiled region) starts with a header line, followed by 381,429 lines of coordinates. When these are combined together into a single two-track BED file, there are a total of 623,736 lines, including the two track header lines. The primary target track header is on line 1, and the capture target header is on line 242307.

Extract Region Coordinates from a One-Track BED File

Use the Linux `wc` and `tail` commands to remove the header line before sorting and merging with BEDtools commands.

Package⇨Tool(s) Used	`wc`
	`tail`
	BEDtools⇨`sort`
	BEDtools⇨`merge`

Input(s)	SAMPLE_RNG_primary.bed (example shown below) SAMPLE_RNG_capture.bed
Output(s)	SAMPLE_primary_coord.bed SAMPLE_capture_coord.bed

Count the Number of Lines in the Input File

```
wc -l SAMPLE_RNG_primary.bed

      242306

      {total_line_count = 242306 lines, including the header}
```

Extract Region Coordinates (remove header line)

```
tail -n 242305 SAMPLE_RNG_primary.bed > SAMPLE_primary_coord_unmerged_unsorted.bed
      {coordinates_line_count = total_line_count - 1}
```

Sort Region Coordinates

```
/path/to/bedtools sort –i SAMPLE_primary_coord_unmerged_unsorted.bed >
SAMPLE_primary_coord_unmerged_sorted.bed
```

Merge Overlapping and Book-Ended Region Coordinates

```
/path/to/bedtools merge –i SAMPLE_primary_coord_unmerged_sorted.bed >
SAMPLE_primary_coord.bed
```

The example shown above is for a one-track primary target BED file. The workflow for a one-track capture target BED file is identical (simply substitute the appropriate file names). The SAMPLE_primary_coord.bed and SAMPLE_capture_coord.bed files are used by several commands, and are also required to generate Picard interval lists.

Extract Primary Target Region Coordinates from a Two-Track BED File

Use the Linux `grep` command to find the location of the track 2 header located just after the last track 1 coordinate line, use the `head` command to extract the entire primary target track including the header (track 1, target region), and finally use the `tail` command to remove the header line. BEDtools commands are then used to sort and merge overlapping and book-ended regions.

Package⇨Tool(s) Used	`grep`
	`head`
	`tail`
	BEDtools⇨`sort`
	BEDtools⇨`merge`

Input(s)	SAMPLE_RNG.bed
Output(s)	SAMPLE_primary_coord.bed

Find Track Header Line Numbers

```
grep -n track SAMPLE_RNG.bed

      1:track name=target_region description="Target Regions"
      242307:track name=tiled_region description="NimbleGen Tiled Regions"

      {track2_header = line 242307}
```

Extract Primary Target Region Coordinates

```
head -242306 SAMPLE_RNG.bed | tail -242305 >
SAMPLE_primary_coord_unmerged_unsorted.bed
   {track1_end = track2_header – 1} {coordinates1_line_count = track2_header – 2}
```

Sort Region Coordinates

```
/path/to/bedtools sort –i SAMPLE_primary_coord_unmerged_unsorted.bed >
SAMPLE_primary_coord_unmerged_sorted.bed
```

Merge Overlapping and Book-Ended Region Coordinates

```
/path/to/bedtools merge –i SAMPLE_primary_coord_unmerged_sorted.bed >
SAMPLE_primary_coord.bed
```

The SAMPLE_primary_coord.bed file is used by several commands, and is also required to generate the Picard target interval list.

Extract Capture Target Region Coordinates from a Two-Track BED File

Use the Linux `grep` command to find the location of the track 2 header, use the `wc` command to count the total number of lines in the BED file in order to calculate the number of capture target coordinate lines, and finally use the `tail` command to extract the capture target without the header line (track 2, tiled region). BEDtools commands are then used to sort and merge overlapping and book-ended regions.

Package⇨Tool(s) Used	`grep`
	`wc`
	`tail`
	BEDtools⇨`sort`
	BEDtools⇨`merge`

Input(s)	SAMPLE_RNG.bed
Output(s)	SAMPLE_capture_coord.bed

Find Track Header Line Numbers

```
grep -n track SAMPLE_RNG.bed

      1:track name=target_region description="Target Regions"
      242307:track name=tiled_region description="NimbleGen Tiled Regions"

      {track2_header = line 242307}
```

Count Number of Lines

```
wc –l SAMPLE_RNG.bed

      623736

      {total_line_count = 623736 lines}
```

Extract Capture Target Region Coordinates

```
tail -381429 SAMPLE_RNG.bed > SAMPLE_capture_coord_unmerged_unsorted.bed
   {coordinates2_line_count = total_line_count – track2_header}
```

Sort Region Coordinates

```
/path/to/bedtools sort –i SAMPLE_capture_coord_unmerged_unsorted.bed >
SAMPLE_capture_coord_unmerged_sorted.bed
```

Merge Overlapping and Book-Ended Region Coordinates

```
/path/to/bedtools merge –i SAMPLE_capture_coord_unmerged_sorted.bed >
SAMPLE_capture_coord.bed
```

The SAMPLE_capture_coord.bed file is used by several commands, and is also required to generate the Picard bait interval list.

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*. If necessary, follow the steps described in *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files* before using the steps described below.

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

Pad Capture Target Regions

```
/path/to/bedtools slop -i SAMPLE_capture_coord.bed -b 100 -g chromosome_sizes.txt >
SAMPLE_capture_coord_padded_unmerged_unsorted.bed
```

Sort Padded Regions

```
/path/to/bedtools sort –i SAMPLE_padded_capture_coord_unmerged_unsorted.bed >
SAMPLE_padded_capture_coord_unmerged_sorted.bed
```

Merge Overlapping and Book-Ended Padded Regions

```
/path/to/bedtools merge –i SAMPLE_padded_capture_coord_unmerged_sorted.bed >
SAMPLE_padded_capture_coord.bed
```

The example shown above is for padding an extracted capture region coordinate file, but a one-track capture region BED file may also be used since BEDtools will exclude the track line from the output file.

In the 'Pad Capture Target Regions' step, 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 bases would be added to both sides of all targets. 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.

Determine Sum Total Size of Regions in a One-Track BED File

The chromosome_sizes.txt file should be as described in *Add Padding to Capture Target*. If necessary, follow the steps described in *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files* before using the step described below.

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

Input(s)	chromosome_sizes.txt SAMPLE_capture_coord.bed (example shown below) or SAMPLE_primary_coord.bed or SAMPLE_RNG_capture.bed or SAMPLE_RNG_primary.bed
Output(s)	{size}

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

The example shown above is for determining the total size of a capture target region coordinate file, but the same command will work with other one-track BED files, if desired.

Count On-Target Reads

If necessary, follow the steps described in *Generate a Sorted, Indexed BAM File*, *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files*, and *Add Padding to Capture Target Regions* before using the step described below.

Package⇨Tool(s) Used	BEDtools⇨`intersect` `wc`
Input(s)	SAMPLE.bam SAMPLE_primary_coord.bed or SAMPLE_capture_coord.bed or SAMPLE_padded_capture_coord.bed
Output(s)	{count}

```
/path/to/bedtools intersect -bed -abam SAMPLE.bam  -b SAMPLE_primary_coord.bed

or

/path/to/bedtools intersect -bed -abam SAMPLE.bam  -b SAMPLE_capture_coord.bed
or

/path/to/bedtools intersect -bed -abam SAMPLE.bam  -b SAMPLE_padded_capture_coord.bed
```

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 (SAMtools)* or *Basic Mapping Metrics (Picard)* for reporting of the total number of mapped, non-duplicate reads.

Calculate Depth of Coverage

If necessary, follow the steps described in *Index a Reference* , *Generate a Sorted, Indexed BAM File*, and *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files* before using the steps described below.

Package⇨Tool(s) Used	GATK⇨`DepthOfCoverage`
Input(s)	ref.fa {indexed} SAMPLE.bam {indexed} SAMPLE_primary_coord.bed or SAMPLE_capture_coord.bed
Output(s)	SAMPLE_gatk_primary_target_coverage.sample_summary or SAMPLE_gatk_capture_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.bam -o SAMPLE_gatk_primary_target
_coverage -L SAMPLE_primary_coord.bed  -ct 1 -ct 10 -ct 20

or

java -Xmx4g -Xms4g -jar /path/to/GATKFramework/GenomeAnalysisTK.jar -T
DepthOfCoverage –R /path/to/ref.fa -I SAMPLE.bam -o
SAMPLE_gatk_capture_target_coverage -L SAMPLE_capture_coord.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:
http://www.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_sting_gatk_walkers_coverage_DepthOfCoverage.html

Call and Filter Genomic Variants

If necessary, follow the steps described in *Index a Reference* , *Generate a Sorted, Indexed BAM File*, and *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files* before using the steps described below. Use the SAMtools `mpileup` command to call variants (SNPs and small indels), and use the `varFilter` command to filter the variants by read depth.

Package⇨Tool(s) Used	SAMtools⇨`mpileup` SAMtools⇨BCFtools>`view` SAMtools⇨vcfutils>`varFilter`
Input(s)	ref.fa {indexed} SAMPLE.bam {indexed}
Output(s)	SAMPLE_vcfutils_var.flt.vcf

Call Genomic Variants

```
/path/to/samtools mpileup -uf /path/to/ref.fa SAMPLE.bam | /path/to/bcftools view -
bvcg - > SAMPLE_samtools_var.raw.bcf
```

Filter Raw Variants

```
/path/to/bcftools view SAMPLE_samtools_var.raw.bcf | /path/to/vcfutils.pl varFilter
-d 12 -D 250 > SAMPLE_vcfutils_var.flt.vcf
```

The `varFilter` command has two options that control the minimum (`-d`) and maximum (`-D`) read depth, with values in this example of '12' and '250', 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.

Additional downstream variant analysis is not covered in this document, but may consist of comparison against known variants for a sample, 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'. If necessary, follow the steps described in *Generate a Sorted, Indexed BAM File* and *Extract Primary and Capture Target Region Coordinates from One-Track or Two-Track BED Files* before using the steps described below.

Create a Picard Target Interval List

Use the SAMtools `view` command to extract the header from a SAMPLE.bam file, use the Linux `gawk` command to extract and format the interval list body from a SAMPLE_primary_coord.bed file, and finally use the `cat` command to concatenate the two pieces together into a properly-formatted interval list.

Package⇨Tool(s) Used	SAMtools⇨`view`
	`cat`
	`gawk`

Input(s)	SAMPLE.bam SAMPLE_primary_coord.bed
Output(s)	SAMPLE_target_intervals.txt

Create a Picard Interval List Header

```
/path/to/samtools view –H SAMPLE.bam > SAMPLE_bam_header.txt
```

Create a Picard Target Interval List Body

```
cat SAMPLE_primary_coord.bed |
gawk '{print $1 "\t" $2+1 "\t" $3 "\t+\tinterval_" NR}' > SAMPLE_target_body.txt
```

Concatenate to Create a Picard Target Interval List

```
cat SAMPLE_bam_header.txt SAMPLE_target_body.txt > SAMPLE_target_intervals.txt
```

The SAMPLE_target_intervals.txt file is used by the Picard `CalculateHsMetrics` command.

Create a Picard Bait Interval List

Use the SAMtools `view` command to extract the header from a SAMPLE.bam file, use the Linux `gawk` command to extract and format the interval list body from a SAMPLE_capture_coord.bed file, and finally use the `cat` command to concatenate the two pieces together into a properly-formatted interval list.

Package⇨Tool(s) Used	SAMtools⇨`view`
	`cat`
	`gawk`

Input(s)	SAMPLE.bam SAMPLE_capture_coord.bed
Output(s)	SAMPLE_bait_intervals.txt

Create a Picard Interval List Header

```
/path/to/samtools view –H SAMPLE.bam > SAMPLE_bam_header.txt
```

Create a Picard Bait Interval List Body

```
cat SAMPLE_capture_coord.bed |
gawk '{print $1 "\t" $2+1 "\t" $3 "\t+\tinterval_" NR}' > SAMPLE_bait_body.txt
```

Concatenate to Create a Picard Bait Interval List

```
cat SAMPLE_bam_header.txt SAMPLE_bait_body.txt > SAMPLE_bait_intervals.txt
```

The SAMPLE_bait_intervals.txt file is used by the Picard `CalculateHsMetrics` command.

Hybrid Selection (HS) Analysis Metrics

The `CalculateHsMetrics` command calculates a number of metrics assessing the quality of targeted enrichment reads. If necessary, follow the steps described in *Index a Reference* , *Generate a Sorted, Indexed BAM File*, and *Create Picard Interval Lists* before using the step described below.

Package⇨Tool(s) Used	Picard⇨`CalculateHsMetrics`
Input(s)	ref.fa {indexed} SAMPLE.bam {indexed} SAMPLE_target_intervals.txt SAMPLE_bait_intervals.txt
Output(s)	SAMPLE_picard_hs_metrics.txt

```
java -Xmx4g -Xms4g -jar /path/to/Picard/CalculateHsMetrics.jar
BAIT_INTERVALS=SAMPLE_bait_intervals.txt
TARGET_INTERVALS=SAMPLE_target_intervals.txt  INPUT=SAMPLE.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 *http://picard.sourceforge.net/picard-metric-definitions.shtml* for a description of output metrics.

3. References

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

- BEDtools: *https://code.google.com/p/bedtools/*
- BWA: *http://bio-bwa.sourceforge.net/*
- FastQC: *http://www.bioinformatics.babraham.ac.uk/projects/fastqc/*
- GATK: *http://www.broadinstitute.org/gatk/*
- IGV: *http://www.broadinstitute.org/igv/*
- Picard: *http://picard.sourceforge.net/*
- SAMtools (including BCFtools and VCFutils): *http://samtools.sourceforge.net/*
- 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.

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

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. 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 100bp expanded to 100bp total to facilitate probe selection. These are equivalent to the **Target intervals** referred to by Picard.

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

Notice to Purchaser
For patent license limitations for individual products please refer to: *www.technical-support.roche.com*.

For life science research only. Not for use in diagnostic procedures.

07187009001 (1) 0314