RTG Core Operations Manual

Release 3.10

Real Time Genomics

Oct 29, 2018

CONTENTS

CHAPTER

ONE

OVERVIEW

This chapter introduces the features, operational options, and installation requirements of the data analysis software from Real Time Genomics.

1.1 Introduction

RTG software enables the development of fast, efficient software pipelines for deep genomic analysis. RTG is built on innovative search technologies and new algorithms designed for processing high volumes of high-throughput sequencing data from different sequencing technology platforms. The RTG sequence search and alignment functions enable read mapping and protein searches with a unique combination of sensitivity and speed.

RTG-based data production pipelines support unprecedented breadth and depth of analysis on genomic data, transforming researcher visibility into DNA sequence analysis and biological investigation. A comprehensive suite of easy-to-integrate data analysis functions increases the productivity of bioinformatics specialists, freeing them to develop analytical solutions that amplify the investigative ability unique to their organization.

RTG software supports a variety of research and medical genomics applications, such as:

- **Medical Genomic Research** – Compare sequence variants and structural variation between normal and disease genomes, or over a disease progression in the same individual to identity causal loci.
- **Personalized Medicine** – Establish reliable, high-throughput processing pipelines that analyze individual human genomes compared to one or more reference genomes. Use RTG software for detection of sequence variants (SNP and indel calling, intersection scripting), as well as structural variation (coverage depth, and copy number variation).
- **Model Organisms and Basic Research** – Utilize RTG mapping and variant detection commands for focused research applications such as metagenomic species identification and frequency, and metabolic pathway analysis. Map microbial communities to generate gapped alignments of both DNA and protein sequence data.
- **Plant Genomics** – Enable investigations of new crop species and variant detection in genetically diverse strains by leveraging RTG's highly sensitive sequence search capabilities for strain and cross-species mapping applications. Flexible sensitivity tuning controls allow investigators to accommodate very high error rates associated with unique combinations of sequencing system error, genome-specific mutation, and aggressive cross-species comparisons.

1.2 RTG software description

RTG software is delivered as a single executable with multiple commands executed through a command line interface (CLI). Commands are delivered in product packages, and for commercial users each command can be independently enabled through a license key.

Usage:

```
rtg COMMAND [OPTIONS] <REQUIRED>
```

RTG software delivers features in four areas:

- **Sequence Search and Alignment** – RTG software uses patented sequence search technology for the rapid production of genomic sequence data. The `map` command implements read mapping and gapped alignment of sequence data against a reference. The `mapx` command searches translated sequence data against a protein database.
- **Data Analysis** – RTG software supports two pipelines for data analysis - variant detection and metagenomics. Purpose-built variant detection pipeline functions include several commands to identify small sequence variants, a `cnv` command to report copy number variation statistics for structural variation, and a `coverage` command to report read depth across a reference.
- **Reporting Options** – Standard result formats and utility commands report results for validation, and ease development of custom scripts for analysis. Scripts that produce publication quality graphics for visualization of data analysis results are available through Real Time Genomics technical support.
- **Data Center Deployment** – RTG software supports typical data center standards for enterprise deployment. RTG provides automated installation and supports industry standard operating environments and data processing systems to help maintain total cost of ownership objectives in enterprise data centers. The RTG software can be run in compute clusters of varying sizes, and commands take advantage of multi-core processors by default.

See also:

For detailed information about RTG command syntax and usage of individual commands, refer to *RTG Command Reference*.

1.3 Sequence search and alignment

RTG software uses an edit-distance alignment score to determine best fit and alignment accuracy.

RTG software includes optimal sensitivity settings for error and mutation rates, plus command line controls and simulation tools that allow investigators to calibrate sensitivity settings for specific data sets. Extensive filtering and reporting options allow complete control over reported alignments, which leads to greater flexibility for downstream analysis functions.

Key functionality of RTG sequence search and alignment includes:

- Read mapping by nucleotide sequence alignment to a reference genome
- Protein database searching by translated nucleotide sequence searches against protein databases
- Sensitivity tuning using parameter options for substitutions, indels, indel lengths, word or step sizes, and alignment scores
- Filtering and reporting ambiguous reads that map to multiple locations
- Benchmarking and optimization using simulation and evaluation commands

RTG mapping commands have the following characteristics:

- Eliminates need for genome indexing
- Aligns sequence reads of any length
- Allows high mismatch levels for increased sensitivity in longer reads
- Allows detection of short indels with single end (SE) or paired end (PE) data
- Can optionally guarantee the mapping of reads with at least a specified number of substitutions and indels
- Supports a wide range of alignment scores

See also:

For detailed information about sequence search and alignment functionality, refer to *Command Reference, map*.

For more information about the RTG integrated software pipeline, refer to *RTG product usage - baseline progressions*

1.3.1 Data formatting

Prior to RTG data production, reference genome and sometimes read data sequence files are typically first converted to the RTG Sequence Data File (SDF) format. This is an efficient storage format optimized for fast retrieval during data processing.

The RTG `format` / `cg2sdf` commands converts sequencing system read and reference genome sequence data into the SDF format. The `format` command accepts source data in standard file formats (such as FASTA / FASTQ / SAM / BAM) and maintains the integrity and consistency of the source data during the conversion to SDF. Similarly, the `cg2sdf` command accepts data in the custom data format used for read data by Complete Genomics, Inc. Read data may be single-end and paired-end reads of fixed or variable length. Sequence data can be formatted as nucleotide or protein.

An SDF is a directory containing a series of files that delineate sequence and quality score information stored in a binary format, along with metadata that describes the original sequencing system data format type:

```
03/19/2010  12:31 PM    <DIR>          .
03/19/2010  12:31 PM    <DIR>          ..
03/19/2010  12:31 PM             5,038 log
03/19/2010  12:31 PM            24,223 mainIndex
03/19/2010  12:31 PM                75 namedata0
03/19/2010  12:31 PM                 8 nameIndex0
03/19/2010  12:31 PM                56 namepointer0
03/19/2010  12:31 PM        23,267,177 seqdata0
03/19/2010  12:31 PM                56 seqpointer0
03/19/2010  12:31 PM                 8 sequenceIndex0
               8 File(s)      23,296,641 bytes
               2 Dir(s)  400,984,870,912 bytes free
```

See also:

For detailed information about formatting sequencing system reads to RTG SDF, refer to *Data Formatting Commands*

1.3.2 Read mapping and alignment

The `map` command implements read mapping and alignment of sequence data against a reference genome, supporting gapped alignments for both single and paired-end reads. The `cgmap` command performs the same function for the gapped, paired-end read data from Complete Genomics, Inc.

A summary of the mapping results is displayed at the command line following execution of the map command, as shown in the paired-end example below:

```
ARM MAPPINGS
    left     right      both
 6650124  6650124 13300248  64.2% mated uniquely (NH = 1)
  186812   186812   373624   1.8% mated ambiguously (NH > 1)
 1538777  1539520  3078297  14.9% unmated uniquely (NH = 1)
   70667    70125   140792   0.7% unmated ambiguously (NH > 1)
       0        0        0   0.0% unmapped due to read frequency (XC = B)
   13624    13946    27570   0.1% unmapped with no matings but too many hits (XC =␣
↪C)
  109720   109765   219485   1.1% unmapped with poor matings (XC = d)
     984     1003     1987   0.0% unmapped with too many matings (XC = e)
  212158   211688   423846   2.0% unmapped with no matings and poor hits (XC = D)
```

```
       0        0        0   0.0% unmapped with no matings and too many good hits␣
↪(XC = E)
 1569609  1569492  3139101  15.2% unmapped with no hits
10352475 10352475 20704950 100.0% total
```

The following display shows the summary output for single end mapped data from the map command

```
READ MAPPINGS

 875007  87.5% mapped uniquely (NH = 1)
  25174   2.5% mapped ambiguously (NH > 1)
     71   0.0% unmapped due to read frequency (XC = B)
  88729   8.9% unmapped with too many hits (XC = C)
   8940   0.9% unmapped with poor hits (XC = D)
      0   0.0% unmapped with too many good hits (XC = E)
   2079   0.2% unmapped with no hits
1000000 100.0% total
```

Read mapping commands also produce HTML summary reports containing more information about mapping results.

1.3.3 Read mapping output files

The map command creates alignment reports in BAM file format and a summary report file named `summary.txt`. There is also a file called `progress` that can be used to monitor overall progress during a run, and a file named `map.log` containing technical information that may be useful for debugging. Alignment reports may be filtered by alignment score, and/or unmapped, unmated, and ambiguous reads (those that map to multiple locations).

When mapping, the output BAM file is named `alignments.bam`. The reads that did not align to the reference will include `XC` attributes in the BAM file that describe why a read did not map.

See also:

For more information about the RTG `map` command, refer to *Command Reference, map*.

For details on RTG extensions to the BAM file format, refer to *SAM/BAM file extensions (RTG map command output)*

1.3.4 Read mapping sensitivity tuning

The RTG map command uses default sensitivity settings that balance mapping percentage and speed requirements. These settings deliver excellent results in most cases, especially in human read sequence data from Illumina runs with error rates of 2% or less.

However, some experiments demand read mapping that accommodates higher machine error, genome mutation, or cross-species comparison. For these situations, the investigator can set various tuning parameters to increase the mapping percentage.

For reads shorter than 64 bp, RTG allows an investigator to select the number of substitutions and indels that the `map` command will "at least" produce. For example, using the `-a` parameter to specify the number of allowed substitutions (i.e., mismatches) at 1, will guarantee that the `map` command finds all alignments with at least 1 substitution.

For reads equal to or longer than 64 base pairs, RTG allows an investigator to modify word and step size parameters related to the index. These parameters are set by default to 18 or half the read length, whichever is smaller. Decreasing the values (using `-w` for word size and `-s` for step size) will increase the percentage of mapped reads at the expense of additional processing time, and in the case of step size, increased memory usage.

The number of mismatches threshold can be altered to increase or decrease the number of mapped reads. Using the `--max-mated-mismatches` parameter for example, an investigator might limit reported alignments to only those at or lower than the given threshold value.

See also:

For more information about the RTG `map` command's sensitivity and tuning parameters, refer to *Command Reference, map*

1.3.5 Protein search

The `mapx` command implements a search of translated nucleotide sequence data against one or more protein databases, with alignment sensitivity adjusted for gaps and mismatches. With `mapx`, an investigator can sort and classify knowns, and identify homologs and novels.

The `mapx` command accepts reads formatted as nucleotide data and a reference database formatted as protein data. In a two-step process, queries that have one or more exact matches of an *k*-mer against the database during the matching phase are then aligned to the subject sequence with a full edit-distance calculation using the BLOSUM62 scoring matrix.

The `mapx` command outputs the statistical significance of matches based on semi-global alignments (globally across query). Reported search results may be modified by a combination of one or more thresholds on % identity, E value, bit score and alignment score. The output results file is similar in construct to that reported by BLASTX.

See also:

For more information about the RTG `mapx` command please refer to *Command Reference, mapx*

1.3.6 Protein search output files

The `mapx` command writes search results and a summary file in a directory specified by the `-o` parameter at the command line. The summary file is named `summary.txt`. There is also a file called `progress` that can be used to monitor overall progress during a run, and a file named `mapx.log` containing technical information that may be useful for debugging.

The protein search results are written to a file named `alignments.tsv.gz`. Each record in this results file, representing a valid search result, is written as tab-separated fields on a single line. The output fields are very similar to those reported by BLASTX.

See also:

For detailed information about the RTG `mapx` command results file format refer to *Mapx output file description*

1.3.7 Protein search sensitivity tuning

The RTG `mapx` command builds a set of indexes from the translated reads and scans each query for matches according to user-specified sensitivity settings. Sensitivity is set with two parameters. The word size (`-w` or `--word`) parameter specifies match length. The mismatches (`-a` or `--mismatches`) parameter specifies the number and placement of *k*-mers across each translated query.

The alignment score threshold can be altered to increase or decrease the number of mapped reads. Using the `--max-mated-score` parameter for example, an investigator might limit reported alignments to only those at or lower than the given threshold value.

See also:

For more information about the RTG `mapx` command's sensitivity and tuning parameters, refer to *Mapx output file description*

1.3.8 Benchmarking and optimization utilities

RTG benchmarking and optimization utilities consist of simulators that generate read and reference genome sequence data, and evaluators that verify the accuracy of sequence search and data analysis functions. Investigators will use these utility commands to evaluate the use of RTG software in various read mapping and data analysis scenarios.

RTG provides several simulators:

- `genomesim` The `genomesim` command generates a reference genome with one or more segments of varying length and a percentage mix of nucleotide values. Use the command to create simulated genomes for benchmarking and evaluation.
- `readsim` / `cgsim` The `readsim` / `cgsim` commands generate synthetic read sequence data from an input reference genome, introducing errors at a specified rate. Use the commands to create simulated read sets for benchmarking and evaluation.
- `popsim`, `samplesim`, `childsim`, `samplereplay`, `denovosim` These variant simulation commands are used to create mutated genomes from a known reference by adding variants. Use these commands to verify accuracy of variant detection analysis software for a particular experiment using different pipeline settings.

Simulated data that is produced in SDF format can be converted into FASTA and FASTQ format sequence files for use with other tools using the `sdf2fasta` and `sdf2fastq` commands respectively.

See also:

For more information about the RTG simulation commands, refer to *Simulation Commands*. Advice is available to ensure best results. Please contact RTG technical support for assistance.

1.4 Variant detection functions

The RTG variant detection pipeline includes commands for both sequence and structural variation detection: `snp`, `family`, `population`, `somatic`, `cnv` and `coverage`. The types of data available for analysis from the RTG software pipeline include: Bayesian sequence variant calling (`snps.vcf`), structural variation analysis (`cnv.ratio`) and alignment coverage depth (`coverage.bed`).

1.4.1 Sequence variation (SNPs, indels and complex variants)

The `snp` command uses a Bayesian probability model to identify and locate single and multiple nucleotide polymorphisms (SNPs and MNPs), indels, and complex sequence variants. The command uses standard BAM format files as input and reports computed posterior scores, base calls, mapping quality, coverage depth, and supporting statistics for all positions and for all variants. The `snp` command may be instructed to run in either haploid or diploid calling mode, and can perform sex-aware calling to automatically switch between haploid and diploid calling according to sex chromosomes specified for your reference species.

The `snp` command calls single nucleotide polymorphisms (SNPs), multiple nucleotide polymorphisms (MNPs), and complex regions from the sorted chromosome-ordered gapped alignment (BAM) files. The `snp` command makes consensus SNP and MNP calls on a diploid organism at every position (homozygous, heterozygous, and equal) in the reference, and calls indels and complex variants of 1-50 bp (depending on input alignments).

At each position in the reference, a base pair determination is made based on statistical analysis of the accumulated read alignments, in accordance with any priors and quality scores. The resulting predictions and accompanying statistics are reported in industry standard VCF format.

The `snps.vcf` output file reports all the called variants. The location and type of the call, the base pairs (reference and called), and a confidence score are present in the `snps.vcf` output file. Additional ancillary statistics in the output describe read alignment evidence that can be used to further evaluate confidence in the variant. Results may be filtered (post variant calling) by posterior scores, coverage depth, or indels, and filtered report results may be integrated with the SNP calls themselves.

See also:

For more information about the SNP output data, refer to *Command Reference, map*, *Command Reference, snp* for syntax, parameters, and usage of the `map` and `snp` commands.

1.4.2 Sequence variation with Mendelian pedigree

The `family` command uses Bayesian analysis and the constraints of Mendelian inheritance to identify single and multiple nucleotide variants in each member of a family group. It will usually yield a better result than running the `snp` command on each individual because the Mendelian constraints help eliminate erroneous calls.

Family calling is restricted to families comprising a mother, father, and one or more sons and daughters. Family members are identified on the command line by sample names matching those used in the input BAM files. The family caller internally assigns the SAM records to the correct family member based on SAM read group information. If available, it automatically makes use of coverage and quality calibration information computed during mapping. It automatically selects the correct haploid/diploid calling depending on the sex of each individual.

The output is a multi-sample VCF file containing a call for each family member whenever any one of the family differs from the reference genome. Each sample reports a computed posterior, base call, and ancillary statistics as per the `snp` command. In addition, there is an overall posterior representing the joint likelihood of the call across all the samples. As with the other variant detection commands, the VCF output includes a filter column containing markers for high-coverage, high-ambiguity, and equivalent calls. It is not guaranteed that the resulting calls will always be Mendelian across the entire family, as *de novo* mutations are also identified and are automatically annotated in the output VCF.

The `population` command extends calling to multiple samples, which may or may not be related according to a supplied pedigree. Mendelian constraints are employed where appropriate, and in cases where many unrelated samples are being called, an iterative expectation-maximization algorithm updates Bayesian priors to give improved accuracy compared to calling samples individually with the `snp` command.

1.4.3 Somatic sequence variation

The `somatic` command uses Bayesian analysis to identify putative cancer explanations in a tumor sample. As with the snp command, it can identify SNPs, MNPs, indels, and complex sequence variants. It operates on two samples, an original sample (assumed to be non-cancerous) and a derived cancerous sample. The derived sample may be a mixture of non-cancerous and cancerous sequence data. The samples are provided to the somatic command in the form of BAM format files with appropriate sample names selected via the read group mechanism.

The somatic caller produces a VCF file detailing putative cancer explanations consisting of computed posterior scores, base calls, and ancillary statistics for both input samples. The somatic caller handles both haploid and diploid sequences and is sex aware. If available, it automatically makes use of coverage and quality calibration information computed during mapping.

By default the `snps.vcf` output file gives each variant called where the original and derived sample differ, together with a confidence. The file is sorted by genomic position. The same statistics reported by the `snp` command per VCF record are listed for both samples. The filter column contains markers for situations of high-coverage, high-ambiguity, and equivalent calls. This column can be used to discard unwanted results in subsequent processing.

1.4.4 Coverage analysis

The `coverage` command reports read depth across a reference genome with smoothing options, and outputs the results in the industry standard BED format. This can used to view histograms of mapped coverage data and gap length distributions.

Use the `coverage` command as a tool to analyze mapping results and determine how much of the genome is covered with mapping alignments, and how many times the same location has been mapped.

Customizable scripts are available for enabling graphical plotting of the coverage results using `gnuplot`.

See also:

For more information about the RTG coverage analysis, refer to *Command reference, coverage*

1.4.5 Copy number variation (CNV) analysis

The `cnv` command identifies and reports copy number statistics that can be used for the investigation of structural variation.

It is used to identify aberrational CNV region(s) or copy number variations in a mapped read. The RTG `cnv` command identifies and reports the copy number variation ratio between two genomes.

The results of CNV detection are output to a BED file format. Customizable scripts are available for enabling graphical plotting of the CNV results using tools such as `gnuplot`.

See also:

For more information about the CNV output data, refer to *Command Reference, cnv*

1.5 Standard input and output file formats

RTG software produces alignment and data analysis results in standard formats to allow pipeline validation and downstream analysis.

Table : Result file formats for validation and downstream analysis

File type	Description and Usage
BAM, SAM	The RTG `map` and `cgmap` commands produces alignment results in the Binary Sequence Alignment/Map (BAM) format: `alignments.bam` or optionally the compressed ASCII (SAM format) equivalent `alignments.sam.gz`. This allows use of familiar pileup viewers for quick visual inspection of alignment results.
TXT	Many RTG commands output summary statistics as ASCII text files.
TSV	Many RTG commands output results in tab separated ASCII text files. These files can typically be loaded directly into a spreadsheet viewing program like Microsoft Excel or Open Office.
BED	Some RTG commands output results in standard BED formats for further analysis and reporting.
PED	Some RTG commands utilize standard PED format text files for supplying sample pedigree and sex information.
VCF	The `snp`, `family`, `population` and `somatic` commands output results in Variant Call Format (VCF) version 4.1.

See also:

For more information about file format extensions, refer to Appendix *RTG output results file descriptions*

1.5.1 SAM/BAM files created by the RTG map command

The Sequence Alignment/Map (SAM/BAM) format (version 1.3) is a well-known standard for listing read alignments against reference sequences.

SAM records list the mapping and alignment data for each read, ordered by chromosome (or other DNA reference sequence) location.

A sample RTG SAM file is shown in the Appendix. It describes the relationship between a read and a reference sequence, including mismatches, insertions and deletions (indels) as determined by the RTG `map` aligner.

Note: RTG mapped alignments are stored in BAM format with RTG read IDs by default. This default can be overridden using the `--read-names` flag or changed after processing using the RTG `samrename` utility to

label the reads with the sequence identifiers from the original source file. For more information, refer to the SAM 1.3 nomenclature and symbols online at: https://samtools.github.io/hts-specs/SAMv1.pdf

RTG has defined several extensions within the standard SAM/BAM format; be sure to review the SAM/BAM format information in *SAM/BAM file extensions (RTG map command output)* of the *Appendix* to this guide for a complete list of all extensions and differences.

By default the RTG `map` command produces output as compressed binary format BAM files but can be set to produce human readable SAM files instead with the `--sam` flag.

1.5.2 Variant caller output files

The Variant Call Format (VCF) is a widely used standard format for storing SNPs, MNPs and indels.

A sample `snps.vcf` file is provided in the Appendix as an example of the output produced by an RTG variant calling run. Each line in a `snps.vcf` output has tab-separated fields and represents a SNP variation calculated from the mapped reads against the reference genome.

Note: RTG variant calls are stored in VCF format (version 4.1). For more information about the VCF format, refer to the specification online at: https://samtools.github.io/hts-specs/VCFv4.1.pdf

RTG employs several extensions within the standard VCF format; be sure to review the VCF format information in *Small-variant VCF output file description* of the *Appendix* to this guide for a complete list of all extensions and differences.

See also:

For more information about file formats, refer to the *Appendix, RTG output results file descriptions*

1.6 Metagenomic analysis functions

The RTG metagenomic analysis pipeline includes commands for sample contamination filtering, estimation of taxon abundances in a sample and finding relationships between samples.

1.6.1 Contamination filtering

The `mapf` command is used for filtering contaminant reads from a sample. It does this by performing alignment of the reads against a reference of known contaminants and producing an output of the reads that did not align successfully. A common use for this is to remove human DNA from a bacterial sample taken from a body site.

1.6.2 Taxon abundance breakdown

The `species` command is used to find the abundances of taxa within a given sample. This is accomplished by analyzing reference genome alignment data made with a metagenomic reference database of known organisms. It produces output in which each taxon is given a fraction representing its abundance in the sample with upper and lower bounds and a value indicating the confidence that the taxon is actually present in the sample. An HTML report allows interactive examination of the abundances at different taxonomic levels.

1.6.3 Sample relationships

The `similarity` command is used to find relationships between sample read sets. It does this by examining *k*-mer word frequencies and the intersections between sets of reads. This results in the output of a similarity matrix, a principal component analysis and nearest neighbor trees in the Newick and phyloXML formats.

1.6.4 Functional protein analysis

The `mapx` command is used to perform a translated nucleotide search of short reads against a reference protein database. This results in an output similar to that reported by BLASTX.

1.7 Pipelines

Included in the RTG release are some pipeline commands which perform simple end-to-end tasks using other RTG commands. These pipelines use mostly default settings for each of the commands called, and are meant as a guideline to building more complex end-to-end pipelines using our tools. The metagenomic pipeline commands are:

- species composition (`composition-meta-pipeline`)
- functional protein analysis (`functional-meta-pipeline`)
- species composition and functional protein analysis (`composition-functional-meta-pipeline`).

For detailed information about individual pipeline commands see *Pipeline Commands*

1.8 Parallel processing

The comparison of genomic variation on a large scale in real time demands parallel processing capability. Parallel processing of gapped alignments and variant detection is recommended by RTG because it significantly reduces wall clock time.

RTG software includes key features that make it easier for a person to prepare a job for parallel processing. First, RTG mapping commands can be performed on a subset of a large file or set of files either by using the `--start-read` and `--end-read` parameters or for commands that do not support this, by using `sdfsplit` to break a large SDF into smaller pieces. Second, the data analysis commands accept multiple alignment files as input from the command. Third, many RTG commands take a `--region` or `--bed-regions` parameter to allow breaking up tasks into pieces across the reference genome.

See also:

See *RTG Command Reference* for command-specific details, *Administration & Capacity Planning* for detailed information about estimating the number of multi-core servers needed (capacity planning), and *Parallel processing approach* for a deeper discussion of compute cluster operations.

1.9 Installation and deployment

RTG is a self-contained tool that sets minimal expectations on the environment in which it is placed. It comes with the application components it needs to execute completely, yet performance can be enhanced with some simple modifications to the deployment configuration. This section provides guidelines for installing and creating an optimal configuration, starting from a typical recommended system.

RTG software pipeline runs in a wide range of computing environments from dual-core processor laptops to compute clusters with racks of dual processor quad core server nodes. However, internal human genome analysis benchmarks suggest the use of six server nodes of the configuration shown in below.

Table : Recommended system requirements

Processor	Intel Core i7-2600
Memory	48 GB RAM DDR3
Disk	5 TB, 7200 RPM (prefer SAS disk)

RTG Software can be run as a Java JAR file, but platform specific wrapper scripts are supplied to provide improved pipeline ergonomics. Instructions for a quick start installation are provided here.

For further information about setting up per-machine configuration files, please see the `README.txt` contained in the distribution zip file (a copy is also included in this manual's appendix).

1.9.1 Quick start instructions

These instructions are intended for an individual to install and operate the RTG software without the need to establish root / administrator privileges.

RTG software is delivered in a compressed zip file, such as: `rtg-core-3.3.zip`. Unzip this file to begin installation.

Linux and Windows distributions include a Java Virtual Machine (JVM) version 1.8 that has undergone quality assurance testing. RTG may be used on other operating systems for which a JVM version 1.8 or higher is available, such as MacOS X or Solaris, by using the 'no-jre' distribution.

RTG for Java is delivered as a Java application accessed via executable wrapper script (`rtg` on UNIX systems, `rtg.bat` on Windows) that allows a user to customize initial memory allocation and other configuration options. It is recommended that these wrapper scripts be used rather than directly executing the Java JAR.

Here are platform-specific instructions for RTG deployment.

Linux/MacOS X:

- Unzip the RTG distribution to the desired location.
- If your installation requires a license file (`rtg-license.txt`), copy the license file provided by Real Time Genomics into the RTG distribution directory.
- **In a terminal, cd to the installation directory and test for success** by entering `./rtg version`
- On MacOS X, depending on your operating system version and configuration regarding unsigned applications, you may encounter the error message:

```
-bash: rtg: /usr/bin/env: bad interpreter: Operation not permitted
```

  If this occurs, you must clear the OS X quarantine attribute with the command:

```
$ xattr -d com.apple.quarantine rtg
```

- The first time rtg is executed you will be prompted with some questions to customize your installation. Follow the prompts.
- Enter `./rtg help` for a list of rtg commands. Help for any individual command is available using the `--help` flag, e.g.: `./rtg format --help`
- By default, RTG software scripts establish a memory space of 90% of the available RAM - this is automatically calculated. One may override this limit in the `rtg.cfg` settings file or on a per-run basis by supplying `RTG_MEM` as an environment variable or as the first program argument, e.g.: `./rtg RTG_MEM=48g map`
- [OPTIONAL] If you will be running RTG on multiple machines and would like to customize settings on a per-machine basis, copy `rtg.cfg` to `/etc/rtg.cfg`, editing per-machine settings appropriately (requires root privileges). An alternative that does not require root privileges is to copy `rtg.cfg` to `rtg.HOSTNAME.cfg`, editing per-machine settings appropriately, where `HOSTNAME` is the short host name output by the command `hostname -s`

Windows:

- Unzip the RTG distribution to the desired location.
- If your installation requires a license, copy the license file provided by Real Time Genomics (`rtg-license.txt`) into the RTG distribution directory.
- Test for success by entering `rtg version` at the command line. The first time RTG is executed you will be prompted with some questions to customize your installation. Follow the prompts.

- Enter `rtg help` for a list of rtg commands. Help for any individual command is available using the `--help` flag, e.g.: `./rtg format --help`
- By default, RTG software scripts establish a memory space of 90% of the available RAM - this is automatically calculated. One may override this limit by setting the `RTG_MEM` variable in the `rtg.bat` script or as an environment variable.

1.9.2 License Management

Commercial distributions of RTG products require the presence of a valid license key file for operation.

The license key file must be located in the same directory as the RTG executable. The license enables the execution of a particular command set for the purchased product(s) and features.

A license key allows flexible use of the RTG package on any node or CPU core.

To view the current license features at the command prompt, enter:

```
$ rtg license
```

See also:

For more data center deployment and instructions for editing scripts, see *Administration & Capacity Planning*.

1.10 Technical assistance and support

For assistance with any technical or conceptual issue that may arise during use of the RTG product, contact Real Time Genomics Technical Support via email at `support@realtimegenomics.com`

In addition, a discussion group is available at: https://groups.google.com/a/realtimegenomics.com/forum/#!forum/rtg-users

A low-traffic announcements-only group is available at: https://groups.google.com/a/realtimegenomics.com/forum/#!forum/rtg-announce

CHAPTER

TWO

RTG COMMAND REFERENCE

This chapter describes RTG commands with a generic description of parameter options and usage. This section also includes expected operation and output results.

2.1 Command line interface (CLI)

RTG is installed as a single executable in any system subdirectory where permissions authorize a particular community of users to run the application. RTG commands are executed through the RTG command-line interface (CLI). Each command has its own set of parameters and options described in this section. The availability of each command may be determined by the RTG license that has been installed. Contact `support@realtimegenomics.com` to discuss changing the set of commands that are enabled by your license.

Results are organized in results directories defined by command parameters and settings. The command line shell environment should include a set of familiar text post-processing tools, such as `grep`, `awk`, or `perl`. Otherwise, no additional applications such as databases or directory services are required.

2.2 RTG command syntax

Usage:

```
rtg COMMAND [OPTIONS] <REQUIRED>
```

To run an RTG command at the command prompt (either DOS window or Unix terminal), type the product name followed by the command and all required and optional parameters. For example:

```
$ rtg format -o human_REF_SDF human_REF.fasta
```

Typically results are written to output files specified with the `-o` option. There is no default filename or filename extension added to commands requiring specification of an output directory or format.

Many times, unfiltered output files are very large; the built-in compression option generates block compressed output files with the `.gz` extension automatically unless the parameter `-Z` or `--no-gzip` is issued with the command.

Many command parameters require user-supplied information of various types, as shown in the following:

Type	Description
DIR, FILE	File or directory name(s)
SDF	Sequence data that has been formatted to SDF
INT	Integer value
FLOAT	Floating point decimal value
STRING	A sequence of characters for comments, filenames, or labels
REGION	A genomic region specification (see below)

Genomic region parameters take one of the following forms:

- sequence_name (e.g.: `chr21`) corresponds to the entirety of the named sequence.
- sequence_name:start (e.g.: `chr21:100000`) corresponds to a single position on the named sequence.
- sequence_name:start-end (e.g.: `chr21:100000-110000`) corresponds to a range that extends from the specified start position to the specified end position (inclusive). The positions are 1-based.
- sequence_name:position+length (e.g.: `chr21:100000+10000`) corresponds to a range that extends from the specified start position that includes the specified number of nucleotides.
- sequence_name:position~padding (e.g.: `chr21:100000~10000`) corresponds to a range that spans the specified position by the specified amount of padding on either side.

To display all parameters and syntax associated with an RTG command, enter the command and type `--help`. For example: all parameters available for the RTG `format` command are displayed when `rtg format --help` is executed, the output of which is shown below.

```
Usage: rtg format [OPTION]... -o SDF FILE+
                  [OPTION]... -o SDF -I FILE
                  [OPTION]... -o SDF -l FILE -r FILE

Converts the contents of sequence data files (FASTA/FASTQ/SAM/BAM) into the RTG
Sequence Data File (SDF) format.

File Input/Output
  -f, --format=FORMAT            format of input. Allowed values are [fasta,
                                 fastq, sam-se, sam-pe, cg-fastq, cg-sam]
                                 (Default is fasta)
  -I, --input-list-file=FILE     file containing a list of input read files (1
                                 per line)
  -l, --left=FILE                left input file for FASTA/FASTQ paired end
                                 data
  -o, --output=SDF               name of output SDF
  -p, --protein                  input is protein. If this option is not
                                 specified, then the input is assumed to
                                 consist of nucleotides
  -q, --quality-format=FORMAT    format of quality data for fastq files (use
                                 sanger for Illumina 1.8+). Allowed values are
                                 [sanger, solexa, illumina]
  -r, --right=FILE               right input file for FASTA/FASTQ paired end
                                 data
      FILE+                      input sequence files. May be specified 0 or
                                 more times

Filtering
      --duster                   treat lower case residues as unknowns
      --exclude=STRING           exclude input sequences based on their name.
                                 If the input sequence contains the specified
                                 string then that sequence is excluded from the
                                 SDF. May be specified 0 or more times
      --select-read-group=STRING when formatting from SAM/BAM input, only
                                 include reads with this read group ID
      --trim-threshold=INT       trim read ends to maximise base quality above
                                 the given threshold

Utility
      --allow-duplicate-names    disable checking for duplicate sequence names
  -h, --help                     print help on command-line flag usage
      --no-names                 do not include name data in the SDF output
      --no-quality               do not include quality data in the SDF output
      --sam-rg=STRING|FILE       file containing a single valid read group SAM
                                 header line or a string in the form
                                 "@RG\tID:READGROUP1\tSM:BACT_SAMPLE\tPL:ILLUMINA"
```

Required parameters are indicated in the usage display; optional parameters are listed immediately below the

usage information in organized categories.

Use the double-dash when typing the full-word command option, as in `--output`:

```
$ rtg format --output human_REF_SDF human_REF.fasta
```

Commonly used command options provide an abbreviated single-character version of a full command parameter, indicated with only a single dash, (Thus `--output` is the same as specifying the command option with the abbreviated character `-o`):

```
$ rtg format -o human_REF human_REF.fasta
```

A set of utility commands are provided through the CLI: `version`, `license`, and `help`. Start with these commands to familiarize yourself with the software.

The `rtg version` command invokes the RTG software and triggers the launch of RTG product commands, options, and utilities:

```
$ rtg version
```

It will display the version of the RTG software installed, RAM requirements, and license expiration, for example:

```
$rtg version
Product: RTG Core 3.5
Core Version: 6236f4e (2014-10-31)
RAM: 40.0GB of 47.0GB RAM can be used by rtg (84%)
License: Expires on 2015-09-30
License location: /home/rtgcustomer/rtg/rtg-license.txt
Contact: support@realtimegenomics.com

Patents / Patents pending:
US: 7,640,256, 13/129,329, 13/681,046, 13/681,215, 13/848,653,
13/925,704, 14/015,295, 13/971,654, 13/971,630, 14/564,810
UK: 1222923.3, 1222921.7, 1304502.6, 1311209.9, 1314888.7, 1314908.3
New Zealand: 626777, 626783, 615491, 614897, 614560
Australia: 2005255348, Singapore: 128254

Citation:
John G. Cleary, Ross Braithwaite, Kurt Gaastra, Brian S. Hilbush, Stuart
Inglis, Sean A. Irvine, Alan Jackson, Richard Littin, Sahar
Nohzadeh-Malakshah, Mehul Rathod, David Ware, Len Trigg, and Francisco
M. De La Vega. "Joint Variant and De Novo Mutation Identification on
Pedigrees from High-Throughput Sequencing Data." Journal of
Computational Biology. June 2014, 21(6): 405-419.
doi:10.1089/cmb.2014.0029.
(c) Real Time Genomics Inc, 2014
```

To see what commands you are licensed to use, type `rtg license`:

```
$rtg license
License: Expires on 2015-03-30
Licensed to: John Doe
License location: /home/rtgcustomer/rtg/rtg-license.txt

    Command name    Licensed?  Release Level

Data formatting:
    format          Licensed   GA
    sdf2fasta       Licensed   GA
    sdf2fastq       Licensed   GA

Utility:
    bgzip           Licensed   GA
```

```
    index           Licensed   GA
    extract         Licensed   GA
    sdfstats        Licensed   GA
    sdfsubset       Licensed   GA
    sdfsubseq       Licensed   GA
    mendelian       Licensed   GA
    vcfstats        Licensed   GA
    vcfmerge        Licensed   GA
    vcffilter       Licensed   GA
    vcfannotate     Licensed   GA
    vcfsubset       Licensed   GA
    vcfeval         Licensed   GA
    pedfilter       Licensed   GA
    pedstats        Licensed   GA
    rocplot         Licensed   GA
    version         Licensed   GA
    license         Licensed   GA
    help            Licensed   GA
```

To display all commands and usage parameters available to use with your license, type `rtg help`:

```
$ rtg help
Usage: rtg COMMAND [OPTION]...
       rtg RTG_MEM=16G COMMAND [OPTION]...  (e.g. to set maximum memory use to 16␣
↪GB)

Type ``rtg help COMMAND`` for help on a specific command. The
following commands are available:

Data formatting:
      format                convert a FASTA file to SDF
      cg2sdf                convert Complete Genomics reads to SDF
      sdf2fasta             convert SDF to FASTA
      sdf2fastq             convert SDF to FASTQ
      sdf2sam               convert SDF to SAM/BAM
Read mapping:
      map                   read mapping
      mapf                  read mapping for filtering purposes
      cgmap                 read mapping for Complete Genomics data
Protein search:
      mapx                  translated protein search
Assembly:
      assemble              assemble reads into long sequences
      addpacbio             add Pacific Biosciences reads to an assembly
Variant detection:
      calibrate             create calibration data from SAM/BAM files
      svprep                prepare SAM/BAM files for sv analysis
      sv                    find structural variants
      discord               detect structural variant breakends using discordant␣
↪reads
      coverage              calculate depth of coverage from SAM/BAM files
      snp                   call variants from SAM/BAM files
      family                call variants for a family following Mendelian␣
↪inheritance
      somatic               call variants for a tumor/normal pair
      population            call variants for multiple potentially-related␣
↪individuals
      lineage               call de novo variants in a cell lineage
      avrbuild              AVR model builder
      avrpredict            run AVR on a VCF file
      cnv                   call CNVs from paired SAM/BAM files
Metagenomics:
      species               estimate species frequency in metagenomic samples
```

```
      similarity              calculate similarity matrix and nearest neighbor tree
Simulation:
      genomesim               generate simulated genome sequence
      cgsim                   generate simulated reads from a sequence
      readsim                 generate simulated reads from a sequence
      readsimeval             evaluate accuracy of mapping simulated reads
      popsim                  generate a VCF containing simulated population␣
↪variants
      samplesim               generate a VCF containing a genotype simulated from a␣
↪population
      childsim                generate a VCF containing a genotype simulated as a␣
↪child of two parents
      denovosim               generate a VCF containing a derived genotype␣
↪containing de novo variants
      samplereplay            generate the genome corresponding to a sample genotype
      cnvsim                  generate a mutated genome by adding CNVs to a template
Utility:
      bgzip                   compress a file using block gzip
      index                   create a tabix index
      extract                 extract data from a tabix indexed file
      sdfstats                print statistics about an SDF
      sdfsplit                split an SDF into multiple parts
      sdfsubset               extract a subset of an SDF into a new SDF
      sdfsubseq               extract a subsequence from an SDF as text
      sam2bam                 convert SAM file to BAM file and create index
      sammerge                merge sorted SAM/BAM files
      samstats                print statistics about a SAM/BAM file
      samrename               rename read id to read name in SAM/BAM files
      mapxrename              rename read id to read name in mapx output files
      mendelian               check a multi-sample VCF for Mendelian consistency
      vcfstats                print statistics from about variants contained within␣
↪a VCF file
      vcfmerge                merge single-sample VCF files into a single multi-
↪sample VCF
      vcffilter               filter records within a VCF file
      vcfannotate             annotate variants within a VCF file
      vcfsubset               create a VCF file containing a subset of the original␣
↪columns
      vcfeval                 evaluate called variants for agreement with a␣
↪baseline variant set
      pedfilter               filter and convert a pedigree file
      pedstats                print information about a pedigree file
      avrstats                print statistics about an AVR model
      rocplot                 plot ROC curves from vcfeval ROC data files
      usageserver             run a local server for collecting RTG command usage␣
↪information
      version                 print version and license information
      license                 print license information for all commands
      help                    print this screen or help for specified command
```

The help command will only list the commands for which you have a license to use.

To display help and syntax information for a specific command from the command line, type the command and then the –help option, as in:

```
$ rtg format --help
```

Note: The following commands are synonymous: `rtg help format` and `rtg format --help`

See also:

Refer to *Installation and deployment* for information about installing the RTG product executable.

2.3 Data Formatting Commands

2.3.1 format

Synopsis:

The `format` command converts the contents of sequence data files (FASTA/FASTQ/SAM/BAM) into the RTG Sequence Data File (SDF) format. This step ensures efficient processing of very large data sets, by organizing the data into multiple binary files within a named directory. The same SDF format is used for storing sequence data, whether it be genomic reference, sequencing reads, protein sequences, etc.

Syntax:

Format one or more files specified from command line into a single SDF:

```
$ rtg format [OPTION] -o SDF FILE+
```

Format one or more files specified in a text file into a single SDF:

```
$ rtg format [OPTION] -o SDF -I FILE
```

Format mate pair reads into a single SDF:

```
$ rtg format [OPTION] -o SDF -l FILE -r FILE
```

Examples:

For FASTA (`.fa`) genome reference data:

```
$ rtg format -o maize_reference maize_chr*.fa
```

For FASTQ (`.fq`) sequence read data:

```
$ rtg format -f fastq -q sanger -o h1_reads -l h1_sample_left.fq -r h1_sample_
↪right.fq
```

Parameters:

File Input/Output		
`-f`	`--format=FORMAT`	The format of the input file(s). Allowed values are [fasta, fastq, fastq-interleaved, sam-se, sam-pe] (Default is fasta).
`-I`	`--input-list-file=FILE`	Specifies a file containing a list of sequence data files (one per line) to be converted into an SDF.
`-l`	`--left=FILE`	The left input file for FASTA/FASTQ paired end data.
`-o`	`--output=SDF`	The name of the output SDF.
`-p`	`--protein`	Set if the input consists of protein. If this option is not specified, then the input is assumed to consist of nucleotides.
`-q`	`--quality-format=FORMAT`	The format of the quality data for fastq format files. (Use sanger for Illumina1.8+). Allowed values are [sanger, solexa, illumina].
`-r`	`--right=FILE`	The right input file for FASTA/FASTQ paired end data.
	`FILE+`	Specifies a sequence data file to be converted into an SDF. May be specified 0 or more times.

Filtering		
	--duster	Treat lower case residues as unknowns.
	--exclude=STRING	Exclude individual input sequences based on their name. If the input sequence name contains the specified string then that sequence is excluded from the SDF. May be specified 0 or more times.
	--select-read-group=STRING	Set to only include only reads with this read group ID when formatting from SAM/BAM files.
	--trim-threshold=INT	Set to trim the read ends to maximise the base quality above the given threshold.

Utility		
	--allow-duplicate-names	Set to disable duplicate name detection.
-h	--help	Prints help on command-line flag usage.
	--no-names	Do not include sequence names in the resulting SDF.
	--no-quality	Do not include sequence quality data in the resulting SDF.
	--sam-rg=STRING|FILE	Specifies a file containing a single valid read group SAM header line or a string in the form @RG\tID:RG1\tSM:G1_SAMP\tPL:ILLUMINA.

Usage:

Formatting takes one or more input data files and creates a single SDF. Specify the type of file to be converted, or allow default to FASTA format. To aggregate multiple input data files, such as when formatting a reference genome consisting of multiple chromosomes, list all files on the command line or use the `--input-list-file` flag to specify a file containing the list of files to process.

For input FASTA and FASTQ files which are compressed, they must have a filename extension of `.gz` (for gzip compressed data) or `.bz2` (for bzip2 compressed data).

When formatting human reference genome data, it is recommended that the resulting SDF be augmented with chromosome reference metadata, in order to enable automatic sex-aware features during mapping and variant calling. The `format` command will automatically recognize several common human reference genomes and install a reference configuration file. If your reference genome is not recognized, a configuration can be manually adapted from one of the examples provided in the RTG distribution and installed in the SDF directory. The reference configuration is described in *RTG reference file format*.

When using FASTQ input files you must specify the quality format being used as one of `sanger`, `solexa` or `illumina`. As of Illumina pipeline version 1.8 and higher, quality values are encoded in Sanger format and so should be formatted using `--quality-format=sanger`. Output from earlier Illumina pipeline versions should be formatted using `--quality-format=illumina` for Illumina pipeline versions starting with 1.3 and before 1.8, or `--quality-format=solexa` for Illumina pipeline versions less than 1.3.

For FASTQ files that represent paired-end read data, indicate each side respectively using the `--left=FILE` and `--right=FILE` flags. Sometimes paired-end reads are represented in a single FASTQ file by interleaving each side of the read. This type of input can be formatted by specifying `fastq-interleaved` as the format type.

The `mapx` command maps translated DNA sequence data against a protein reference. You must use the `-p`, `--protein` flag to format the protein reference used by `mapx`.

Use the `sam-se` format for single end SAM/BAM input files and the `sam-pe` format for paired end SAM/BAM input files. Note that if the input SAM/BAM files are sorted in coordinate order (for example if they have already been aligned to a reference), it is recommended that they be shuffled before formatting, so that subsequent mapping is not biased by processing reads in chromosome order. For example, a BAM file can be shuffled using `samtools collate` as follows:

```
$ samtools collate -uOn 256 reads.bam tmp-prefix >reads_shuffled.bam
```

And this can be carried out on the fly during formatting using bash process redirection in order to reduce intermediate I/O, for example:

```
$ rtg format --format sam-pe <(samtools collate -uOn 256 reads.bam temp-prefix) ...
```

The SDF for a read set can contain a SAM read group which will be automatically picked up from the input SAM/BAM files if they contain only one read group. If the input SAM/BAM files contain multiple read groups you must select a single read group from the SAM/BAM file to format using the `--select-read-group` flag or specify a custom read group with the `--sam-rg` flag. The `--sam-rg` flag can also be used to add read group information to reads given in other input formats. The SAM read group stored in an SDF will be automatically used during mapping the reads it contains to provide tracking information in the output BAM files.

The `--trim-threshold` flag can be used to trim poor quality read ends from the input reads by inspecting base qualities from FASTQ input. If and only if the quality of the final base of the read is less than the threshold given, a new read length is found which maximizes the overall quality of the retained bases using the following formula.

$$\arg\max x \left(\sum_{i=x+1}^{l} (T - q(i)) \right) \text{ if } q(l) < T$$

Where *l* is the original read length, *x* is the new read length, *T* is the given threshold quality and *q(n)* is the quality of the base at the position n of the read.

Note: Sequencing system read files and reference genome files often have the same extension and it may not always be obvious which file is a read set and which is a genome. Before formatting a sequencing system file, open it to see which type of file it is. For example:

```
$ less pf3.fa
```

In general, a read file typically begins with an `@` or + character; a genome reference file typically begins with the characters `chr`.

Normally when the input data contains multiple sequences with the same name the format command will fail with an error. The `--allow-duplicate-names` flag will disable this check conserving memory, but if the input data has multiple sequences with the same name you will not be warned. Having duplicate sequence names can cause problems with other commands, especially for reference data since the output many commands identifies sequences by their names.

See also:

sdf2fasta, *sdf2fastq*, *sdfstats*

2.3.2 cg2sdf

Synopsis:

Converts Complete Genomics sequencing system reads to RTG SDF format.

Syntax:

Multi-file input specified from command line:

```
$ rtg cg2sdf [OPTION]... -o SDF FILE+
```

Multi-file input specified in a text file:

```
$ rtg cg2sdf [OPTION]... -o SDF -I FILE
```

Example:

```
$ rtg cg2sdf -I CG_source_files -o CG_reads
```

Parameters:

File Input/Output		
-I	--input-list-file=FILE	File containing a list of Complete Genomics TSV files (1 per line)
-o	--output=SDF	Name of output SDF.
	FILE+	File in Complete Genomics TSV format. May be specified 0 or more times.

Filtering		
	--max-unknowns=INT	Maximum number of Ns allowed in either side for a read (Default is 5)

Utility		
-h	--help	Print help on command-line flag usage.
	--no-quality	Does not include quality data in the resulting SDF.
	--sam-rg=STRING|FILE	File containing a single valid read group SAM header line or a string in the form @RG\tID:RG1\tSM:G1_SAMP\tPL:COMPLETE.

Usage:

The cg2sdf command converts Complete Genomics reads into an RTG SDF.

RTG supports two versions of Complete Genomics reads: the original 35 bp paired end read structure ("version 1"); and the newer 29 bp paired end structure ("version 2"). The 29 bp reads are sometimes equivalently represented as 30 bp with a redundant single base overlap containing an 'N' at position 20. This alternate representation is automatically normalised by RTG during processing.

The command accepts input files in the Complete Genomics read data format entered at the command line. The reads for a single sample are typically supplied in a large number of files. For consistent operation with multiple samples, use the -I, --input-list-file flag to specify a text file that lists all the files to format, specifying one filename per line.

Using the --sam-rg flag the SDF for a read set can contain the SAM read group specified. The SAM read group stored in an SDF will be automatically used during mapping the reads it contains to provide tracking information in the output BAM files. For version 1 reads, the platform (PL) must be specified as COMPLETE, and for version 2 reads, the platform must be specified as COMPLETEGENOMICS.

Complete Genomics often produces "no calls" in the reads, represented by multiple Ns. Sometimes, numerous Ns indicate a low quality read. The --max-unknowns option limits how many Ns will be added to the SDF during conversion. If there are more than the specified number of Ns in one arm of the read, they read will not be added to the SDF.

See also:

format, *sdf2cg*, *cgmap*, *sdf2fasta*, *sdf2fastq*, *sdfstats*, *sdfsplit*

2.3.3 sdf2cg

Synopsis:

Converts SDF formatted data into Complete Genomics TSV file(s).

Syntax:

Extract specific sequences listed on command line:

```
$ rtg sdf2cg [OPTION]... -i SDF -o FILE STRING+
```

Extract specific sequences listed in text file

```
$ rtg sdf2cg [OPTION]... -i SDF -o FILE -I FILE
```

Extract range of sequences by sequence id

```
$ rtg sdf2cg [OPTION]... -i SDF -o FILE --end-id INT --start-id INT
```

Parameters:

File Input/Output		
-i	--input=SDF	SDF containing sequences
-o	--output=FILE	Output filename (extension added if not present). Use '-' to write to standard output

Filtering		
	--end-id=INT	Exclusive upper bound on sequence id
-I	--id-file=FILE	File containing sequence ids, or sequence names if –names flag is set, one per line
-n	--names	Interpret supplied sequence as names instead of numeric ids
	--start-id=INT	Inclusive lower bound on sequence id
	STRING+	ID of sequence to extract, or sequence name if –names flag is set. May be specified 0 or more times

Utility		
-h	--help	Print help on command-line flag usage
-l	--line-length=INT	Maximum number of nucleotides to print on a line of output. A value of 0 indicates no limit (Default is 0)
-Z	--no-gzip	Do not gzip the output

Usage:

The `sdf2cg` command converts RTG SDF data into Complete Genomics reads format.

While any SDF data can be consumed by this command to produce a CG TSV file, real Complete Genomics data typically has specific read lengths and other characteristics that would make normal data fed through this command inappropriate for use in a Complete Genomics pipeline. However this command can be used to turn SDF formatted CG data back into TSV close to its original form.

See also:

cg2sdf

2.3.4 sdf2fasta

Synopsis:

Convert SDF data into a FASTA file.

Syntax:

```
$ rtg sdf2fasta [OPTION]... -i SDF -o FILE
```

Example:

```
$ rtg sdf2fasta -i humanSDF -o humanFASTA_return
```

Parameters:

File Input/Output		
-i	--input=SDF	SDF containing sequences.
-o	--output=FILE	Output filename (extension added if not present). Use '-' to write to standard output.

Filtering		
	`--end-id=INT`	Only output sequences with sequence id less than the given number. (Sequence ids start at 0).
	`--start-id=INT`	Only output sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).
`-I`	`--id-file=FILE`	Name of a file containing a list of sequences to extract, one per line.
	`--names`	Interpret any specified sequence as names instead of numeric sequence ids.
	`--taxons`	Interpret any specified sequence as taxon ids instead of numeric sequence ids. This option only applies to a metagenomic reference species SDF.
	`STRING+`	Specify one or more explicit sequences to extract, as sequence id, or sequence name if –names flag is set.

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
	`--interleave`	Interleave paired data into a single output file. Default is to split to separate output files.
`-l`	`--line-length=INT`	Set the maximum number of nucleotides or amino acids to print on a line of FASTA output. Should be nonnegative, with a value of 0 indicating that the line length is not capped. (Default is 0).
`-Z`	`--no-gzip`	Set this flag to create the FASTA output file without compression. By default the output file is compressed with blocked gzip.

Usage:

Use the `sdf2fasta` command to convert SDF data into FASTA format. By default, `sdf2fasta` creates a separate line of FASTA output for each sequence. These lines will be as long as the sequences themselves. To make them more readable, use the `-l`, `--line-length` flag and define a reasonable record length like 75.

By default all sequences will be extracted, but flags may be specified to extract reads within a range, or explicitly specified reads (either by numeric sequence id or by sequence name if `--names` is set). Additionally, when the input SDF is a metagenomic species reference SDF, the `--taxons` option, any supplied id is interpreted as a taxon id and all sequences assigned directly to that taxon id will be output. This provides a convenient way to extract all sequence data corresponding to a single (or multiple) species from a metagenomic species reference SDF.

Sequence ids are numbered starting at 0, the `--start-id` flag is an inclusive lower bound on id and the `--end-id` flag is an exclusive upper bound. For example if you have an SDF with five sequences (ids: 0, 1, 2, 3, 4) the following command:

```
$ rtg sdf2fasta --start-id=3 -i mySDF -o output
```

will extract sequences with id 3 and 4. The command:

```
$ rtg sdf2fasta --end-id=3 -i mySDF -o output
```

will extract sequences with id 0, 1, and 2. And the command:

```
$ rtg sdf2fasta --start-id=2 --end-id=4 -i mySDF -o output
```

will extract sequences with id 2 and 3.

See also:

format, *sdf2fastq*, *sdfstats*

2.3.5 sdf2fastq

Synopsis:

Convert SDF data into a FASTQ file.

Syntax:

```
$ rtg sdf2fastq [OPTION]... -i SDF -o FILE
```

Example:

```
$ rtg sdf2fastq -i humanSDF -o humanFASTQ_return
```

Parameters:

File Input/Output		
-i	--input=SDF	Specifies the SDF data to be converted.
-o	--output=FILE	Specifies the file name used to write the resulting FASTQ output.

Filtering		
	--end-id=INT	Only output sequences with sequence id less than the given number. (Sequence ids start at 0).
	--start-id=INT	Only output sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).
-I	--id-file=FILE	Name of a file containing a list of sequences to extract, one per line.
	--names	Interpret any specified sequence as names instead of numeric sequence ids.
	STRING+	Specify one or more explicit sequences to extract, as sequence id, or sequence name if –names flag is set.

Utility		
-h	--help	Prints help on command-line flag usage.
-q	--default-qualty=INT	Set the default quality to use if the SDF does not contain sequence quality data (0-63).
	--interleave	Interleave paired data into a single output file. Default is to split to separate output files.
-l	--line-length=INT	Set the maximum number of nucleotides or amino acids to print on a line of FASTQ output. Should be nonnegative, with a value of 0 indicating that the line length is not capped. (Default is 0).
-Z	--no-gzip	Set this flag to create the FASTQ output file without compression. By default the output file is compressed with blocked gzip.

Usage:

Use the `sdf2fastq` command to convert SDF data into FASTQ format. If no quality data is available in the SDF, use the `-q`, `--default-quality` flag to set a quality score for the FASTQ output. The quality encoding used during output is sanger quality encoding. By default, `sdf2fastq` creates a separate line of FASTQ output for each sequence. As with `sdf2fasta`, there is an option to use the `-l`, `--line-length` flag to restrict the line lengths to improve readability of long sequences.

By default all sequences will be extracted, but flags may be specified to extract reads within a range, or explicitly specified reads (either by numeric sequence id or by sequence name if `--names` is set).

It may be preferable to extract data to unaligned SAM/BAM format using `sdf2sam`, as this preserves read-group information stored in the SDF and may also be more convenient when dealing with paired-end data.

The `--start-id` and `--end-id` flags behave as in `sdf2fasta`.

See also:

format, *sdf2fasta*, *sdf2sam*, *sdfstats*

2.3.6 sdf2sam

Synopsis:

Convert SDF read data into unaligned SAM or BAM format file.

Syntax:

```
$ rtg sdf2sam [OPTION]... -i SDF -o FILE
```

Example:

```
$ rtg sdf2sam -i samplereadsSDF -o samplereads.bam
```

Parameters:

File Input/Output		
-i	--input=SDF	Specifies the SDF data to be converted.
-o	--output=FILE	Specifies the file name used to write the resulting SAM/BAM to. The output format is automatically determined based on the filename specified. If '-' is given, the data is written as uncompressed SAM to standard output.

Filtering		
	--end-id=INT	Only output sequences with sequence id less than the given number. (Sequence ids start at 0).
	--start-id=INT	Only output sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).
-I	--id-file=FILE	Name of a file containing a list of sequences to extract, one per line.
	--names	Interpret any specified sequence as names instead of numeric sequence ids.
	STRING+	Specify one or more explicit sequences to extract, as sequence id, or sequence name if –names flag is set.

Utility		
-h	--help	Prints help on command-line flag usage.
-Z	--no-gzip	Set this flag when creating SAM format output to disable compression. By default SAM is compressed with blocked gzip, and BAM is always compressed.

Usage:

Use the `sdf2sam` command to convert SDF data into unaligned SAM/BAM format. By default all sequences will be extracted, but flags may be specified to extract reads within a range, or explicitly specified reads (either by numeric sequence id or by sequence name if `--names` is set). This command is a useful way to export paired-end data to a single output file while retaining any read group information that may be stored in the SDF.

The output format is either SAM/BAM depending on the specified output file name. e.g. `output.sam` or `output.sam.gz` will output as SAM, whereas `output.bam` will output as BAM. If neither SAM or BAM format is indicated by the file name then BAM will be used and the output file name adjusted accordingly. e.g `output` will become `output.bam`. However if standard output is selected (-) then the output will always be in uncompressed SAM format.

The `--start-id` and `--end-if` behave as in `sdf2fasta`.

See also:

format, *sdf2fasta*, *sdf2fastq*, *sdfstats*, *cg2sdf*, *sdfsplit*

2.3.7 fastqtrim

Synopsis:

Trim reads in FASTQ files.

Syntax:

```
$ rtg fastqtrim [OPTION]... -i FILE -o FILE
```

Example:

Apply hard base removal from the start of the read and quality-based trimming of terminal bases:

```
$ rtg fastqtrim -s 12 -E 18 -i S12_R1.fastq.gz -o S12_trimmed_R1.fastq.gz
```

Parameters:

File Input/Output		
-i	--input=FILE	Input FASTQ file, Use '-' to read from standard input.
-o	--output=FILE	Output filename. Use '-' to write to standard output.
-q	--quality-format=FORMAT	Quality data encoding method used in FASTQ input files (Illumina 1.8+ uses sanger). Allowed values are [sanger, solexa, illumina] (Default is sanger)

Filtering		
	--discard-empty-reads	Discard reads that have zero length after trimming. Should not be used with paired-end data.
-E	--end-quality-threshold=INT	Trim read ends to maximise base quality above the given threshold (Default is 0)
	--min-read-length=INT	If a read ends up shorter than this threshold it will be trimmed to zero length (Default is 0)
-S	--start-quality-threshold=INT	Trim read starts to maximise base quality above the given threshold (Default is 0)
-e	--trim-end-bases=INT	Always trim the specified number of bases from read end (Default is 0)
-s	--trim-start-bases=INT	Always trim the specified number of bases from read start (Default is 0)

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
-r	--reverse-complement	If set, output in reverse complement.
	--seed=INT	Seed used during subsampling.
	--subsample=FLOAT	If set, subsample the input to retain this fraction of reads.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

Use `fastqtrim` to apply custom trimming and preprocessing to raw FASTQ files prior to mapping and alignment. The `format` command contains some limited trimming options, which are applied to all input files, however in some cases different or specific trimming operations need to be applied to the various input files. For example, for paired-end data, different trimming may need to be applied for the left read files compared to the right read files. In these cases, `fastqtrim` should be used to process the FASTQ files first.

The `--end-quality-threshold` flag can be used to trim poor quality bases from the ends of the input reads by inspecting base qualities from FASTQ input. If and only if the quality of the final base of the read is less than the threshold given, a new read length is found which maximizes the overall quality of the retained bases using the following formula:

$$\arg\max x \left(\sum_{i=x+1}^{l} (T - q(i)) \right) \text{ if } q(l) < T$$

where *l* is the original read length, *x* is the new read length, *T* is the given threshold quality and *q(n)* is the quality of the base at the position *n* of the read. Similarly, `--start-quality-threshold` can be used to apply this quality-based thresholding to the start of reads.

Some of the trimming options may result in reads that have no bases remaining. By default, these are output as zero-length FASTQ reads, which RTG commands are able to handle normally. It is also possible to remove zero-length reads altogether from the output with the `--discard-empty-reads` option, however this should not be used when processing FASTQ files corresponding to paired-end data, otherwise the pairs in the two files will no longer be matched.

Similarly, when using the `--subsample` option to down-sample a FASTQ file for paired-end data, you should specify an explicit randomization seed via `--seed` and use the same seed value for the left and right files.

Formatting with filtering on the fly

Running custom filtering with `fastqtrim` need not mean that additional disk space is required or that formatting be slowed down due to additional disk I/O. It is possible when using standard unix shells to perform the filtering on the fly. The following example demonstrates how to apply different trimming options to left and right files while formatting to SDF:

```
$ rtg format -f fastq -o S12_trimmed.sdf \
    -l <(rtg fastqtrim -s 12 -E 18 -i S12_R1.fastq.gz -o -)
    -r <(rtg fastqtrim -E 18 -i S12_R2.fastq.gz -o -)
```

See also:

format

2.3.8 petrim

Synopsis:

Trim paired-end read FASTQ files based on read arm alignment overlap.

Syntax:

```
$ rtg petrim [OPTION]... -l FILE -o FILE -r FILE
```

Parameters:

File Input/Output		
`-l`	`--left=FILE`	Left input FASTQ file (AKA R1)
`-o`	`--output=FILE`	Output filename prefix. Use '-' to write to standard output.
`-q`	`--quality-format=FORMAT`	Quality data encoding method used in FASTQ input files (Illumina 1.8+ uses sanger). Allowed values are [sanger, solexa, illumina] (Default is sanger)
`-r`	`--right=FILE`	Right input FASTQ file (AKA R2)

Sensitivity Tuning		
	`--aligner-band-width=FLOAT`	Aligner indel band width scaling factor, fraction of read length allowed as an indel (Default is 0.5)
	`--gap-extend-penalty=INT`	Penalty for a gap extension during alignment (Default is 1)
	`--gap-open-penalty=INT`	Penalty for a gap open during alignment (Default is 19)
`-P`	`--min-identity=INT`	Minimum percent identity in overlap to trigger overlap trimming (Default is 90)
`-L`	`--min-overlap-length=INT`	Minimum number of bases in overlap to trigger overlap trimming (Default is 25)
	`--mismatch-penalty=INT`	Penalty for a mismatch during alignment (Default is 9)
	`--soft-clip-distance=INT`	Soft clip alignments if indels occur INT bp from either end (Default is 5)
	`--unknowns-penalty=INT`	Penalty for unknown nucleotides during alignment (Default is 5)

Filtering		
	--discard-empty-pairs	If set, discard pairs where both reads have zero length (after any trimming)
	--discard-empty-reads	If set, discard pairs where either read has zero length (after any trimming)
	--left-probe-length=INT	Assume R1 starts with probes this long, and trim R2 bases that overlap into this (Default is 0)
-M	--midpoint-merge	If set, merge overlapping reads at midpoint of overlap region. Result is in R1 (R2 will be empty)
-m	--midpoint-trim	If set, trim overlapping reads to midpoint of overlap region.
	--min-read-length=INT	If a read ends up shorter than this threshold it will be trimmed to zero length (Default is 0)
	--mismatch-adjustment=STRING	Method used to alter bases/qualities at mismatches within overlap region. Allowed values are [none, zero-phred, pick-best] (Default is none)
	--right-probe-length=INT	Assume R2 starts with probes this long, and trim R1 bases that overlap into this (Default is 0)

Utility		
-h	--help	Print help on command-line flag usage.
	--interleave	Interleave paired data into a single output file. Default is to split to separate output files.
-Z	--no-gzip	Do not gzip the output.
	--seed=INT	Seed used during subsampling.
	--subsample=FLOAT	If set, subsample the input to retain this fraction of reads.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

Paired-end read sequencing with read lengths that are long relative to the typical library fragment size can often result in the same bases being sequenced by both arms. This repeated sequencing of bases within the same fragment can skew variant calling, and so it can be advantageous to remove such read overlap.

In some cases, complete read-through can occur, resulting in additional adaptor or non-genomic bases being present at the ends of reads.

In addition, some library preparation methods rely on the ligation of synthetic probe sequence to attract target DNA, which is subsequently sequenced. Since these probe bases do not represent genomic material, they must be removed at some point during the analytic pipeline prior to variant calling, otherwise they could act as a reference bias when calling variants. Removal from the primary arm where the probe is attached is typically easy enough (e.g. via `fastqtrim`), however in cases of high read overlap, probe sequence can also be present in the other read arm.

`petrim` aligns each read arm against it's mate with high stringency in order to identify cases of read overlap. The sensitivity of read overlap detection is primarily controlled through the use of `--min-identity` and `--min-overlap-length`, although it is also possible to adjust the penalties used during alignment.

The following types of trimming or merging may be applied.

- Removal of non-genomic bases due to complete read-through. This removal is always applied.
- Removal of overlap bases impinging into regions occupied by probe bases. For example, if the left arms contain 11-mer probes, using `--left-probe-length=11` will result in the removal of any right arm bases that overlap into the first 11 bases of the left arm. Similar trimming is available for situations where probes are ligated to the right arm by using `--right-probe-length`.
- Adjustment of mismatching read bases inside areas of overlap. Such mismatches indicate that one or other of the bases has been incorrectly sequenced. Alteration of these bases is selected by supplying the `--mismatch-adjustment` flag with a value of `zero-phred` to alter the phred quality score of both bases to zero, or `pick-best` to choose whichever base had the higher reported quality score.

- Removal of overlap regions by trimming both arms back to a point where no overlap is present. An equal number of bases are removed from each arm. This trimming is enabled by specifying `--midpoint-trim` and takes place after any read-through or probe related trimming.
- Merging non-redundant sequence from both reads to create a single read, enabled via `--midpoint-merge`. This is like `--midpoint-trim` with a subsequent moving of the R2 read onto the end of the the R1 read (thus the R2 read becomes empty).

After trimming or merging it is possible that one or both of the arms of the pair have no bases remaining, and a strategy is needed to handle these pairs. The default is to retain such pairs in the output, even if one or both are zero-length. When both arms are zero-length, the pair can be dropped from output with the use of `--discard-empty-pairs`. If downstream processing cannot handle zero-length reads, `--discard-empty-reads` will drop a read pair if either of the arms is zero-length.

`petrim` also provides the ability to down-sample a read set by using the `--subsample` option. This will produce a different sampling each time, unless an explicit randomization seed is specified via `--seed`.

Formatting with paired-end trimming on the fly

Running custom filtering with `petrim` can be done in standard Unix shells without incurring the use of additional disk space or unduly slowing down the formatting of reads. The following example demonstrates how to apply paired-end trimming while formatting to SDF:

```
$ rtg format -f fastq-interleaved -o S12_trimmed.sdf \
    <(rtg petrim -l S12_R1.fastq.gz -r S12_R2.fastq.gz -m -o - --interleaved)
```

This can even be combined with `fastqtrim` to provide extremely flexible trimming:

```
$ rtg format -f fastq-interleaved -o S12_trimmed.sdf \
    <(rtg petrim -m -o - --interleave \
         -l <(rtg fastqtrim -s 12 -E 18 -i S12_R1.fastq.gz -o -) \
         -r <(rtg fastqtrim -E 18 -i S12_R2.fastq.gz -o -) \
     )
```

Note: `petrim` currently assumes Illumina paired-end sequencing, and aligns the reads in FR orientation. Sequencing methods which produce arms in a different orientation can be processed by first converting the input files using `fastqtrim --reverse-complement`, running `petrim`, followed by another `fastqtrim --reverse-complement` to restore the reads to their original orientation.

See also:

fastqtrim, *format*

2.4 Read Mapping Commands

2.4.1 map

Synopsis:

The `map` command aligns sequence reads onto a reference genome, creating an alignments file in the Sequence Alignment/Map (SAM) format. It can be used to process single-end or paired-end reads, of equal or variable length.

Syntax:

Map using an SDF or a single end sequence file:

```
$ rtg map [OPTION]... -o DIR -t SDF -i SDF|FILE
```

Map using paired end sequence files:

```
$ rtg map [OPTION]... -o DIR -t SDF -l FILE -r FILE
```

Example:

```
$ rtg map -t strain_REF -i strain_READS -o strain_MAP -b 2 -U
```

Parameters:

File Input/Output		
-F	--format=FORMAT	Input format for reads. Allowed values are [sdf, fasta, fastq, sam-se, sam-pe] (Default is sdf)
-i	--input=SDF|FILE	Input read set.
-l	--left=FILE	Left input file for FASTA/FASTQ paired end reads.
-o	--output=DIR	Directory for output.
-q	--quality-format=FORMAT	Quality data encoding method used in FASTQ input files (Illumina 1.8+ uses sanger). Allowed values are [sanger, solexa, illumina] (Default is sanger)
-r	--right=FILE	Right input file for FASTA/FASTQ paired end reads.
	--sam	Output the alignment files in SAM format.
-t	--template=SDF	SDF containing template to map against.

Sensitivity Tuning		
	`--aligner-band-width=FLOAT`	Set the fraction of the read length that is allowed to be an indel. Decreasing this factor will allow faster processing, at the expense of only allowing shorter indels to be aligned. (Default is 0.5).
	`--aligner-mode=STRING`	Set the aligner mode to be used. Allowed values are [auto, table, general] (Default is auto).
	`--bed-regions=FILE`	Restrict calibration to mappings falling within the regions in the supplied BED file.
	`--blacklist-threshold=INT`	filter k-mers that occur more than this many times in the reference using a blacklist
	`--gap-extend-penalty=INT`	Set the penalty for extending a gap during alignment. (Default is 1).
	`--gap-open-penalty=INT`	Set the penalty for a gap open during alignment. (Default is 19).
`-c`	`--indel-length=INT`	Guarantees number of positions that will be detected in a single indel. For example, -c 3 specifies 3 nucleotide insertions or deletions. (Default is 1).
`-b`	`--indels=INT`	Guarantees minimum number of indels which will be detected when used with read less than 64 bp long. For example -b 1 specifies 1 insertion or deletion. (Default is 1).
`-M`	`--max-fragment-size=INT`	The maximum permitted fragment size when mating paired reads. (Default is 1000).
`-m`	`--min-fragment-size=INT`	The minimum permitted fragment size when mating paired reads. (Default is 0).
	`--mismatch-penalty=INT`	Set the penalty for a mismatch during alignment. (Default is 9).
`-d`	`--orientation=STRING`	Set the orientation required for proper pairs. Allowed values are [fr, rf, tandem, any] (Default is any).
	`--pedigree=FILE`	Genome relationships pedigree containing sex of sample.
	`--repeat-freq=INT%`	Where INT specifies the percentage of all hashes to keep, discarding the remaining percentage of the most frequent hashes. Increasing this value will improve the ability to map sequences in repetitive regions at a cost of run time. It is also possible to specify the option as an absolute count (by omitting the percent symbol) where any hash exceeding the threshold will be discarded from the index. (Default is 90%).
	`--sex=SEX`	Specifies the sex of the individual. Allowed values are [male, female, either].
	`--soft-clip-distance=INT`	Set to soft clip alignments when an indel occurs within that many nucleotides from either end of the read. (Default is 5).
`-s`	`--step=INT`	Set the step size. (Default is word size).
`-a`	`--substitutions=INT`	Guarantees minimum number of substitutions to be detected when used with read data less than 64 bp long. (Default is 1).
	`--unknowns-penalty=INT`	Set the penalty for unknown nucleotides during alignment. (Default is 5).
`-w`	`--word=INT`	Specifies an internal minimum word size used during the initial matching phase. Word size selection optimizes the number of reads for a desired level of sensitivity (allowed mismatches and indels) given an acceptable alignment speed. (Default is 22, or read length / 2, whichever is smaller).

Filtering		
	`--end-read=INT`	Only map sequences with sequence id less than the given number. (Sequence ids start at 0).
	`--start-read=INT`	Only map sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).

Reporting		
	`--all-hits`	Output all alignments meeting thresholds instead of applying mating and N limits.
	`--max-mated-mismatches=INT`	The maximum mismatches for mappings across mated results, alias for `--max-mismatches` (as absolute value or percentage of read length). (Default is 10%).
`-e`	`--max-mismatches=INT`	The maximum mismatches for mappings in single-end mode (as absolute value or percentage of read length). (Default is 10%).
`-n`	`--max-top-results=INT`	Sets the maximum number of reported mapping results (locations) per read when it maps to multiple locations with the same alignment score (AS). Allowed values are between 1 and 255. (Default is 5).
`-E`	`--max-unmated-mismatches=INT`	The maximum mismatches for mappings of unmated results (as absolute value or percentage of read length). (Default is 10%).
	`--sam-rg=STRING|FILE`	Specifies a file containing a single valid read group SAM header line or a string in the form `@RG\tID:RG1\tSM:BACT\tPL:ILLUMINA`.
	`--top-random`	If set, will only output a single random top hit for each read.

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
	`--legacy-cigars`	Produce cigars in legacy format (using M instead of X or =) in SAM/BAM output. When set will also produce the MD field.
	`--no-calibration`	Set this flag to not produce the calibration output files.
`-Z`	`--no-gzip`	Set this flag to create the SAM output files without compression. By default the output files are compressed with tabix compatible blocked gzip.
	`--no-merge`	Set to output mated, unmated and unmapped alignment records into separate SAM/BAM files.
	`--no-svprep`	Do not perform structural variant processing.
	`--no-unmapped`	Do not output unmapped reads. Some reads that map multiple times will not be aligned, and are reported as unmapped. These reads are reported with XC attributes that indicate the reason they were not mapped.
	`--no-unmated`	Do not output unmated reads when in paired-end mode.
	`--read-names`	Output read names instead of sequence ids in SAM/BAM files. (Uses more RAM).
	`--tempdir=DIR`	Set the directory to use for temporary files during processing. (Defaults to output directory).
`-T`	`--threads=INT`	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

The `map` command locates reads against a reference using an indexed search method, aligns reads at each location, and then reports alignments within a threshold as a record in a BAM file. Some extensions have been made to the output format. Please consult *SAM/BAM file extensions (RTG map command output)* for more information.

By default the alignment records will be output into a single BAM format file called `alignments.bam`. When the `--sam` flag is set it will instead be output in compressed SAM format to a file called `alignments.sam.gz`.

When using the `--no-merge` flag the output will be put into separate files for mated, unmated and unmapped records depending on the kind of reads being mapped. When mapping single end reads it will produce a single

output file containing the mappings called `alignments.bam`. When mapping paired end reads it will produce two files, `mated.bam` with paired alignments and `unmated.bam` with unpaired alignments. A file containing the unmapped reads called `unmapped.bam` is also produced in both cases. When used in conjunction with the `--sam` flag each of the separate files will be in compressed SAM format rather than BAM format.

It is highly recommended to ensure that read group tracking information is present in the output BAM files. When mapping directly from a SAM/BAM file with a single read group, or from an SDF with the read group information stored this is automatically set and does not need to be set manually. This read group information can also be explicitly supplied by using the `--sam-rg` flag to provide a SAM-formatted read group header line. The header should specify at least the `ID`, `SM` and `PL` fields for the read group. For more details see *Using SAM/BAM Read Groups in RTG map*.

During mapping RTG automatically creates a calibration file alongside each BAM file containing information about base qualities, average coverage levels etc. This calibration information is utilized during variant calling to give more accurate results and to determine when coverage levels are abnormally high. When processing exome data, it is important that this calibration information should only be computed for mappings within the exome capture regions, using the `--bed-regions` flag to give the name of a bed file containing your vendor-supplied exome capture regions, otherwise the computed coverage levels will be much lower than actual and subsequent variant calling will be affected. Calibration computation is disabled when read group information is not present. If you decide to merge BAM files, it is recommended that you use the `sammerge` command, as this is aware of the calibration files and will ensure that the calibration information is preserved through the merge process. Calibration information can also be explicitly regenerated for a BAM file by using the `calibrate` command.

Alignments are measured with an alignment score where each match adds 0, each mismatch (substitution) adds `--mismatch-penalty` (default 9), each gap (insertion or deletion) adds `--gap-open-penalty` (default 19), and each gap extension adds `--gap-extend-penalty` (default 1). For more information about alignment scoring see *Alignment scoring*.

The `--aligner-band-width` parameter controls the size of indels that can be aligned. It represents the fraction of the read length that can be aligned as an indel. Decreasing this factor will allow faster processing, at the expense of only allowing shorter indels to be aligned.

The `--aligner-mode` parameter controls which aligner is used during mapping. The `table` setting uses an aligner that constrains alignments to those containing at most one insertion or deletion and uses an in-built non-affine penalty table (this is not currently user modifiable) with different penalties for insertions vs deletions of various lengths. This allows for faster alignment and better identification of longer indels. The `general` setting will use the same aligner as previous versions of RTG. The default `auto` setting will choose the table aligner when mapping Illumina data (as determined by the `PLATFORM` field of the SAM read group supplied) and the general aligner otherwise.

As indels near the ends of reads are not necessarily very accurate, the `--soft-clip-distance` parameter is used to set when soft clipping should be employed at read ends. If an indel is found within the distance specified from either end of the read, the bases leading to the indel from that end and the indel itself will be soft clipped instead.

The number of mismatches threshold is set with the `-e` parameter (`--max-mismatches`) as either an absolute value or a percentage of the read length.

The `map` command accepts formatted reference genome and read data in the sequence data format (SDF), which is generated with the format command. Sequences can be of any length.

The `map` command delivers reliable results with all sensitivity tuning and number of mismatches defaults. However, investigators can optimize mapping percentages with minimal introduction of noise (i.e., false positive alignments) by adjusting sensitivity settings.

For all read lengths, increasing the number of mismatches threshold percentage will pick up additional reads that haven't mapped as well to the reference. Take this approach when working with high error rates introduced by genome mutation or cross-species mapping.

For reads under 64 base pairs in length, setting the `-a` (`--substitutions`), `-b` (`--indels`), and `-c` (`--indel-length`) options will guarantee mapping of reads with at least the specified number of nucleotide substitutions and gaps respectively. Think of it as a floor rather than a ceiling, as all reads will be aligned within the number of mismatches threshold. Some of these alignments could have more substitutions (or more gaps and longer gap lengths) but still score within the threshold.

For reads equal to or greater than 64 base pairs in length, adjust the word and step size by setting the `-w` (`--word`) and `-s` (`--step`) options, respectively. RTG `map` is a hash-based alignment algorithm and the word flag defines the length of the hash used. Indexes are created for the read sequence data with each `map` command instance, which allows the flexible tuning.

Decreasing the word size increases the percentage mapped against the trade-off of time. Small word size reductions can deliver a material difference in mapping with minimal introduction of noise. Decreasing the step size increases the percentage mapped incrementally, but requires some more time and a cost of higher memory consumption. In both cases, the trade-offs get more severe as you get farther away from the default settings and closer to the percentage mapped maximum.

Another important parameter to consider is the `--repeat-freq` flag, which allows a trade-off to be made between run time and ability to map repetitive reads. When repetitive data is present, a relatively small proportion of the data can account for much of the run time, due to the large number of potential mapping locations. By discarding the most repetitive hashes during index building, we can dramatically reduce elapsed run time, without affecting the mapping of less-repetitive reads. There are two mechanisms by which this trade off can be controlled. The `--repeat-freq` flag accepts an integer that denotes the frequency at which hashes will be discarded. For example, `--repeat-freq=20` will discard all hashes that occur 20 or more times in the index. Alternatively specify a percentage of total hashes to retain in the index, discarding most repetitive hashes first. For example `--repeat-freq=95%` will discard up to the most frequent 5% of hashes. Using a percentage based threshold is recommended, as this yields a more consistent trade off as the size of a data set varies, which is important when investigating appropriate flag settings on a subset of the data before embarking on large-scale mapping, or when performing mapping on a cluster of servers using a variety of read set sizes. The default value has been selected to provide a balance between speed and accuracy when mapping human whole genome sequencing reads against a non-repeat-masked reference.

An alternative to `--repeat-freq` is the `--blacklist-threshold` flag. When set it completely overrides the behaviour controlled by the `--repeat-freq` flag, instead using the threshold specified against a blacklist installed in the reference SDF (an error will be reported if an appropriate blacklist is not available for the selected `--word` size). The concept is similar to `--repeat-freq` except the hashes to exclude are based off frequency within the reference rather than within the read set, this is most useful when the read data is high coverage targeted data. This option doesn't support the % based threshold, however since the thresholding is based off the reference values are portable against different read set sizes. A blacklist can be created/installed using the *hashdist* command.

Some reads will map to the reference more than once with the same alignment score. These ambiguous reads may add noise that reduces the accuracy of SNP calling, or increase the available information for copy number variation reporting in structural variation analysis. Rather than throw this information away, or make an arbitrary decision about the read, the RTG `map` command identifies all locations where a read maps and provides parameters to show or hide such alignments at varying thresholds. Parameter sweeps are typically used to determine the optimal settings that maximize percent mapped. If in doubt, contact RTG technical support for advice.

Some reads which are marked as unmapped did have potential placements but didn't meet some other criteria, these unmapped records are annotated with an XC code, you can check the *SAM/BAM file extensions (RTG map command output)* to find out what these codes mean.

When using the `--legacy-cigars` flag we also output a MD attribute on SAM records to enable location of mismatches.

When the sex of the individual being mapped is specified using the `--pedigree` or `--sex` flag the reference genome SDF must contain a `reference.txt` reference configuration file. For details of how to construct a reference text file see *RTG reference file format*.

When running many copies of `map` in parallel on different samples within a larger project, special consideration should be made with respect to where the data resides. Reading and writing data from and to a single disk partition may result in undesirable I/O performance characteristics. To help alleviate this use the `--tempdir` flag to specify a separate disk partition for temporary files and arrange for inputs and outputs to reside on separate disk partitions where possible. For more details see *Task 4 - Map reads to the reference genome*.

See also:

format, *calibrate*, *cgmap*, *mapf*, *mapx hashdist*

2.4.2 mapf

Synopsis:

Filters reads for contaminant sequences by mapping them against the contaminant reference. It outputs two SDF files, one containing the input reads that map to the reference and one that contains those that do not.

Syntax:

Filter an SDF or other single-file sequence source:

```
$ rtg mapf [OPTION]... -o DIR -t SDF -i SDF|FILE
```

Filter paired end sequence files:

```
$ rtg mapf [OPTION]... -o DIR -t SDF -l FILE -r FILE
```

Example:

```
$ rtg mapf -i reads -o filtered -t sequences
```

Parameters:

File Input/Output		
	--bam	Output the alignment files in BAM format.
-F	--format=FORMAT	Input format for reads. Allowed values are [sdf, fasta, fastq, sam-se, sam-pe] (Default is sdf)
-i	--input=SDF|FILE	Input read set.
-l	--left=FILE	Left input file for FASTA/FASTQ paired end reads.
-o	--output=DIR	Directory for output.
-q	--quality-format=FORMAT	Quality data encoding method used in FASTQ input files (Illumina 1.8+ uses sanger). Allowed values are [sanger, solexa, illumina] (Default is sanger)
-r	--right=FILE	Right input file for FASTA/FASTQ paired end reads.
	--sam	Output the alignment files in SAM format.
-t	--template=SDF	SDF containing template to map against.

Sensitivity Tuning		
	`--aligner-band-width=FLOAT`	Set the fraction of the read length that is allowed to be an indel. Decreasing this factor will allow faster processing, at the expense of only allowing shorter indels to be aligned. (Default is 0.5).
	`--aligner-mode=STRING`	Set the aligner mode to be used. Allowed values are [auto, table, general] (Default is auto).
	`--blacklist-threshold=INT`	filter k-mers that occur more than this many times in the reference using a blacklist
	`--gap-extend-penalty=INT`	Set the penalty for extending a gap during alignment. (Default is 1).
	`--gap-open-penalty=INT`	Set the penalty for a gap open during alignment. (Default is 19).
`-c`	`--indel-length=INT`	Guarantees number of positions that will be detected in a single indel. For example, -c 3 specifies 3 nucleotide insertions or deletions. (Default is 1).
`-b`	`--indels=INT`	Guarantees minimum number of indels which will be detected when used with read less than 64 bp long. For example -b 1 specifies 1 insertion or deletion. (Default is 1).
`-M`	`--max-fragment-size=INT`	The maximum permitted fragment size when mating paired reads. (Default is 1000).
`-m`	`--min-fragment-size=INT`	The minimum permitted fragment size when mating paired reads. (Default is 0).
	`--mismatch-penalty=INT`	Set the penalty for a mismatch during alignment. (Default is 9).
`-d`	`--orientation=STRING`	Set the orientation required for proper pairs. Allowed values are [fr, rf, tandem, any] (Default is any).
	`--pedigree=FILE`	Genome relationships pedigree containing sex of sample.
	`--repeat-freq=INT%`	Where INT specifies the percentage of all hashes to keep, discarding the remaining percentage of the most frequent hashes. Increasing this value will improve the ability to map sequences in repetitive regions at a cost of run time. It is also possible to specify the option as an absolute count (by omitting the percent symbol) where any hash exceeding the threshold will be discarded from the index. (Default is 90%).
	`--sex=SEX`	Specifies the sex of the individual. Allowed values are [male, female, either].
	`--soft-clip-distance=INT`	Set to soft clip alignments when an indel occurs within that many nucleotides from either end of the read. (Default is 5).
`-s`	`--step=INT`	Set the step size. (Default is half word size).
`-a`	`--substitutions=INT`	Guarantees minimum number of substitutions to be detected when used with read data less than 64 bp long. (Default is 1).
	`--unknowns-penalty=INT`	Set the penalty for unknown nucleotides during alignment. (Default is 5).
`-w`	`--word=INT`	Specifies an internal minimum word size used during the initial matching phase. Word size selection optimizes the number of reads for a desired level of sensitivity (allowed mismatches and indels) given an acceptable alignment speed. (Default is 22).

Filtering		
	`--end-read=INT`	Exclusive upper bound on read id.
	`--start-read=INT`	Inclusive lower bound on read id.

Reporting		
	`--max-mated-mismatches=INT`	Maximum mismatches for mappings across mated results, alias for –max-mismatches (as absolute value or percentage of read length) (Default is 10%)
`-e`	`--max-mismatches=INT`	Maximum mismatches for mappings in single-end mode (as absolute value or percentage of read length) (Default is 10%)
`-E`	`--max-unmated-mismatches=INT`	Maximum mismatches for mappings of unmated results (as absolute value or percentage of read length) (Default is 10%)
	`--sam-rg=STRING|FILE`	File containing a single valid read group SAM header line or a string in the form `@RG\tID:READGROUP1\tSM:BACT_SAMPLE\tPL:ILLUMINA`

Utility		
`-h`	`--help`	Print help on command-line flag usage.
	`--legacy-cigars`	Use legacy cigars in output.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--no-merge`	Output mated/unmated/unmapped alignments into separate SAM/BAM files.
	`--read-names`	Use read name in output instead of read id (Uses more RAM)
	`--tempdir=DIR`	Directory used for temporary files (Defaults to output directory)
`-T`	`--threads=INT`	Number of threads (Default is the number of available cores)

Usage:

Use to filter out contaminant reads based on a set of possible contaminant sequences. The command maps the reads against the provided contaminant sequences and produces two SDF output files, one which contains the sequences which mapped to the contaminant and one which contains the sequences which did not. The SDF which contains the unmapped sequences can then be used as input to further processes having had the contaminant reads filtered out.

This command differs from regular `map` in that paired-end read arms are kept together – on the assumption that it does not make sense from a contamination viewpoint that one arm came from the contaminant genome and the other did not. Thus, with `mapf`, if either end of the read maps to the contaminant database, both arms of the read are filtered.

Note: The `--sam-rg` flag specifies the read group information when outputting to SAM/BAM and also adjusts the internal aligner configuration based on the platform given. Recognized platforms are `ILLUMINA`, `LS454`, and `IONTORRENT`.

See also:

map, *cgmap*, *mapx*

2.4.3 cgmap

Synopsis:

Mapping function for Complete Genomics data.

Syntax:

```
$ rtg cgmap [OPTION]... -i SDF|FILE --mask STRING -o DIR -t SDF
```

Example:

```
$ rtg cgmap -i CG_reads --mask cg1 -o CG_map -t HUMAN_reference
```

Parameters:

File Input/Output		
-F	--format=FORMAT	Format of read data. Allowed values are [sdf, tsv] (Default is sdf)
-i	--input=SDF|FILE	Specifies the Complete Genomics reads to be mapped.
-o	--output=DIR	Specifies the directory where results are reported.
	--sam	Set to output results in SAM format instead of BAM format.
-t	--template=SDF	Specifies the SDF containing the reference genome to map against.

Sensitivity Tuning		
	--blacklist-threshold=INT	filter k-mers that occur more than this many times in the reference using a blacklist
	--mask=STRING	Read indexing method. Allowed values are [cg1, cg1-fast, cg2]
-M	--max-fragment-size=INT	The maximum permitted fragment size when mating paired reads. (Default is 1000).
-m	--min-fragment-size=INT	The minimum permitted fragment size when mating paired reads. (Default is 0).
-d	--orientation=STRING	Orientation for proper pairs. Allowed values are [fr, rf, tandem, any] (Default is any)
	--pedigree	Genome relationships pedigree containing sex of sample.
	--penalize-unknowns	If set, will treat unknown bases as mismatches.
	--repeat-freq=INT%	Where INT specifies the percentage of all hashes to keep, discarding the remaining percentage of the most frequent hashes. Increasing this value will improve the ability to map sequences in repetitive regions at a cost of run time. It is also possible to specify the option as an absolute count (by omitting the percent symbol) where any hash exceeding the threshold will be discarded from the index. (Default is 95%).
	--sex=SEX	Sex of individual. Allowed values are [male, female, either]

Filtering		
	--end-read=INT	Only map sequences with sequence id less than the given number. (Sequence ids start at 0).
	--start-read=INT	Only map sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).

Reporting		
	--all-hits	Output all alignments meeting thresholds instead of applying mating and N limits.
-e	--max-mated-mismatches=INT	The maximum mismatches allowed for mated results (as absolute value or percentage of read length). (Default is 10%).
-n	--max-top-results=INT	Sets the maximum number of reported mapping results (locations) with the same alignment score (AS). Allowed values are between 1 and 255. (Default is 5).
-E	--max-unmated-mismatches=INT	The maximum mismatches allowed for unmated results (as absolute value or percentage of read length). (Default is 10%).
	--sam-rg=STRING|FILE	Specifies a file containing a single valid read group SAM header line or a string in the form @RG\tID:RG1\tSM:BACT\tPL:COMPLETE.

Utility		
-h	--help	Prints help on command-line flag usage.
	--legacy-cigars	Produce cigars in legacy format (using M instead of X or =) in SAM/BAM output. When set will also produce the MD field.
	--no-calibration	Set this flag to not produce the calibration output files.
-Z	--no-gzip	Set this flag to create the SAM output files without compression. By default the output files are compressed with tabix compatible blocked gzip.
	--no-merge	Set to output mated, unmated and unmapped alignment records into separate SAM/BAM files.
	--no-svprep	Do not perform structural variant processing.
	--no-unmapped	Do not output unmapped reads. Some reads that map multiple times will not be aligned, and are reported as unmapped. These reads are reported with XC attributes that indicate the reason they were not mapped.
	--no-unmated	Do not output unmated reads when in paired-end mode.
	--tempdir=DIR	Set the directory to use for temporary files during processing. (Defaults to output directory).
-T	--threads=INT	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

The `cgmap` command is similar in functionality to the map command with some key differences for mapping the unique structure of Complete Genomics reads.

RTG supports two versions of Complete Genomics reads: the original 35 bp paired end read structure ("version 1"); and the newer 29 bp paired end structure ("version 2"). The 29 bp reads are sometimes equivalently represented as 30 bp with a redundant single base overlap containing an *N* at position 20. This alternate representation is automatically normalised by RTG during processing.

When specifying SAM read group information during mapping, the platform should be set according to the read structure. For version 1 reads, the platform (`PL`) must be specified as `COMPLETE`, and for version 2 reads, the platform must be specified as `COMPLETEGENOMICS`.

Where the `map` command allows you to control the mapping sensitivity using the substitutions (`-a`), indels (`-b`) and indel lengths (`-c`) flags, `cgmap` provides presets using the `--mask` flag. You will need to select a mask that is appropriate for the version of reads you are mapping. For version 1 the mask `cg1-fast` is approximately equivalent to setting the substitutions to 1 and indels to 1 in the `map` command, whereas the mask `cg1` provides more sensitivity to substitutions (somewhere between 1 and 2). For version 2 the mask `cg2` is approximately equivalent to the mask `cg1`.

See also:

map, *mapf*, *mapx*

2.4.4 coverage

Synopsis:

The `coverage` command measures and reports coverage depth of read alignments across a reference.

Syntax:

Multi-file input specified from command line:

```
$ rtg coverage [OPTION]... -o FILE+
```

Multi-file input specified in a text file:

```
$ rtg coverage [OPTION]... -o DIR -I FILE
```

Example:

```
$ rtg coverage -o h1_coverage alignments.bam
```

Parameters:

File Input/Output		
	--bed-regions=FILE	If set, only read SAM records that overlap the ranges contained in the specified BED file.
	--bedgraph	If set, output in BEDGRAPH format (suppresses BED file output)
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
-o	--output=DIR	Directory for output.
	--per-base	If set, output per-base counts in TSV format (suppresses BED file output)
	--per-region	If set, output BED/BEDGRAPH entries per-region rather than every coverage level change.
	--region=REGION	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
-t	--template=SDF	SDF containing the reference genome.
	FILE+	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	--exclude-mated	Exclude all mated SAM records.
	--exclude-unmated	Exclude all unmated SAM records.
	--keep-duplicates	Don't detect and filter duplicate reads based on mapping position.
-m	--max-as-mated=INT	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
-u	--max-as-unmated=INT	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
-c	--max-hits=INT	If set, ignore SAM records with an alignment count that exceeds this value.
	--min-mapq=INT	If set, ignore SAM records with MAPQ less than this value.
-s	--smoothing=INT	Smooth with this number of neighboring values (0 means no smoothing) (Default is 50)

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

The `coverage` command calculates coverage depth by counting all alignments from input SAM/BAM files against a specified reference genome. Sensitivity tuning parameters allow the investigator to test and identify the most appropriate set of alignments to use in downstream analysis.

The `coverage` command provides insight into sequencing coverage for each of the reference sequences. Use to validate mapping results and determine how much of the reference is covered with alignments and how many times the same location is mapped. Gaps indicate no coverage in a specific location.

The default output of coverage will create a new BED entry whenever the coverage level changes. The `--smoothing` flag may be supplied to smooth over a number of neighboring values in order to reduce noise and variation in the output coverage data file. Typical values range from 0-100 but there is no limit.

When the average coverage levels over specific regions is of interest, specify the `--per-region` option. Rather than creating a new coverage entry when the coverage level changes, this mode will output one record for each region of interest containing the average coverage statistics within the region.

For detailed information on the coverage levels at a per-base resolution is available by using the `--per-base` option, but be aware that the output files can be very large, so this is of most use when focusing on particular regions..

See also:

map, *snp*, *cnv*

2.4.5 calibrate

Synopsis:

Creates quality calibration files for all supplied SAM/BAM files.

Syntax:

Multi-file input specified from command line:

```
$ rtg calibrate [OPTION]... -t SDF FILE+
```

Multi-file input specified in a text file:

```
$ rtg calibrate [OPTION]... -t SDF -I FILE
```

Example:

```
$ rtg calibrate -t hs_reference hs_map/alignments.bam
```

Parameters:

File Input/Output		
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-m`	`--merge=FILE`	If set, merge records and calibration files to this output file.
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	`--bed-regions=FILE`	Restrict calibration to mappings falling within the supplied BED regions.
	`--exclude-bed=FILE`	BED containing regions to exclude from calibration.
	`--exclude-vcf=FILE`	VCF containing sites of known variants to exclude from calibration.

Utility		
`-f`	`--force`	Force overwriting of calibration files.
`-h`	`--help`	Print help on command-line flag usage.
`-T`	`--threads=INT`	Number of threads (Default is the number of available cores)

Usage:

Use to create quality calibration files for existing SAM/BAM mapping files which can be used in later commands to improve results. The calibration file will have .calibration appended to the SAM/BAM file name. If the `--merge` option is used, this command can be used to simultaneously merge input files to a single, calibrated output file.

See also:

snp, *map*, *cgmap*

2.5 Protein Search Commands

2.5.1 mapx

Synopsis:

The RTG `mapx` command searches translated read data sets of defined length (e.g., 100 bp reads) against protein databases or translated nucleotide sequences. This similarity search with gapped alignment may be adjusted for sensitivity to gaps and mismatches. Reported search results may be modified by a combination of one or more thresholds on % identity, E value, bit score and alignment score. The output file of the command is similar to that reported by BLASTX.

Syntax:

```
$ rtg mapx [OPTION]... -i SDF|FILE -o DIR -t SDF
```

Example:

```
$ rtg mapx -i SDF_reads -o DIR_Mappings -t SDF_proteinRef
```

Parameters:

File Input/Output		
`-F`	`--format=FORMAT`	Input format for reads. Allowed values are [sdf, fasta, fastq, sam-se] (Default is sdf)
`-i`	`--input=SDF|FILE`	Query read sequences.
`-o`	`--output=DIR`	Directory for output.
`-t`	`--template=SDF`	SDF containing protein database to search.

Sensitivity Tuning		
`-c`	`--gap-length=INT`	Guarantees number of positions that will be detected in a single gap. (Default is 1).
`-b`	`--gaps=INT`	Guarantees minimum number of gaps which will be detected (if this is larger than the minimum number of mismatches then the minimum number of mismatches is increased to the same value). (Default is 0).
	`--matrix=STRING`	The name of the scoring matrix used during alignment. Allowed values are [blosum45, blosum62, blosum80] (Default is blosum62).
	`--min-dna-read-length=INT`	Specifies the minimum read length in nucleotides. Shorter reads will be ignored. (Defaults to 3 * (word-size + mismatches + 1)).
`-a`	`--mismatches=INT`	Guarantees minimum number of identical mismatches which will be detected. (Default is 1).
	`--repeat-freq=INT%`	Where INT specifies the percentage of all hashes to keep, discarding the remaining percentage of the most frequent hashes. Increasing this value will improve the ability to map sequences in repetitive regions at a cost of run time. It is also possible to specify the option as an absolute count (by omitting the percent symbol) where any hash exceeding the threshold will be discarded from the index. (Default is 95%).
`-w`	`--word=INT`	Specifies an internal minimum word size used during the initial matching phase. Word size selection optimizes the number of reads for a desired level of sensitivity (allowed mismatches and gaps) given an acceptable alignment speed. (Default is 7).

Filtering		
	`--end-read=INT`	Exclusive upper bound on read id.
	`--start-read=INT`	Inclusive lower bound on read id.

Reporting		
	`--all-hits`	Output all alignments meeting thresholds instead of applying topn/topequals N limits.
`-e`	`--max-alignment-score=INT`	The maximum alignment score at output (as absolute value or percentage of read length in protein space). (Default is 30%).
`-E`	`--max-e-score=FLOAT`	The maximum e-score at output. (Default is 10.0).
`-n`	`--max-top-results=INT`	Sets the maximum number of reported mapping results (locations) per read when it maps to multiple locations with the same alignment score (AS). Allowed values are between 1 and 255. (Default is 10).
`-B`	`--min-bit-score=FLOAT`	The minimum bit score at output.
`-P`	`--min-identity=INT`	The minimum percent identity at output. (Default is 60).
`-f`	`--output-filter=STRING`	The output filter. Allowed values are [topequal, topn] (Default is topn).

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
`-Z`	`--no-gzip`	Set this flag to create the output files without compression. By default the output files are compressed with blocked gzip.
	`--no-unmapped`	Do not output unmapped reads. Some reads that map multiple times will not be aligned and can be optionally reported as unmapped in a separate unmapped.tsv file.
	`--read-names`	Output read names instead of sequence ids in output files. (Uses more RAM)
	`--suppress-protein`	Suppresses the output of sequence protein information.
	`--tempdir=DIR`	Set the directory used for temporary files during processing. (Defaults to output directory).
`-T`	`--threads=INT`	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

Use the `mapx` command for translated nucleotide search against a protein database. The command outputs the statistical significance of matches based on semi-global alignments (globally across query), in contrast to local alignments reported by BLAST.

This command requires a protein reference instead of a DNA reference. When formatting the protein reference, use the `-p` (`--protein`) flag in the format command.

The `mapx` command builds a set of indexes from the translated reads and scans each database query for matches according to user-specified sensitivity settings. Sensitivity is set with two parameters: the word size (`-w`) parameter that governs match length and the mismatches (`-a`) parameter that governs the number and placement of *k*-mers across each translated query.

Mapping large read sets may require more RAM than is available on the current hardware. In these cases, mapping can be split into smaller jobs each mapping a subset of the reads, where the range of reads to map is specified using the `--start-read` and `--end-read` flags.

The formation of an index group with `-w` and `-a` combinations permits the guaranteed return of all query-subject matches where the non-matching residue count is equal to or less than the `-a` setting. Higher levels of mismatches are typically detected but not explicitly guaranteed.

In a two-step matching and alignment process, queries that have one or more exact matches of an *k*-mer against the database during the matching phase are then aligned to the subject sequence. The alignment algorithm employs a full edit-distance calculation using the BLOSUM62 scoring matrix. Resulting alignment can be filtered on E value, bit score, % identity or raw alignment score.

The `mapx` command generates a tabular results file called `alignments.tsv` in the output directory. This ASCII file contains columns of reported data in a format similar to that produced by BLASTX.

See also:

map, *cgmap*, *mapf*

2.6 Assembly Commands

2.6.1 assemble

Synopsis:

The `assemble` command combines short reads into longer contigs. It first constructs a de Bruijn graph and then maps those reads/read pairs into the graph in order to resolve ambiguities. The reads must be converted to RTG SDF format with the `format` command prior to assembly.

Syntax:

Assemble a set of Illumina reads into long contigs, and then use read mappings to resolve ambiguities:

```
$ rtg assemble [OPTION] --consensus-reads INT -k INT -o DIR SDF+
```

Assemble a set of 454 reads into long contigs:

```
$ rtg assemble [OPTION] --consensus-reads INT -k INT -o DIR -f SDF
```

Assemble a set of 454 and Illumina reads at once:

```
$ rtg assemble [OPTION] --consensus-reads INT -k INT -o DIR ILLUMINA_SDF -f 454_SDF
```

Improve an existing assembly by mapping additional reads and attempting to improve the consensus:

```
$ rtg assemble [OPTION] -g GRAPH-DIR -k INT -o DIR SDF
```

Example:

Illumina reads:

```
$ rtg assemble --consensus-reads 7 -k 32 -o assembled Illumina_reads.sdf \
  --alignments assembly.alignments
```

Combining Illumina and 454 reads:

```
$ rtg assemble --consensus-reads 7 -k 32 -o assembled Illumina_reads.sdf \
  -f 454_reads.sdf
```

Parameters:

File Input/Output		
`-f`	`--454=DIR`	SDF containing 454 sequences to assemble. May be specified 0 or more times.
`-g`	`--graph=DIR`	If you have already constructed an assembly and would like to map additional reads to it or apply some alternate filters you can use this flag to specify the existing graph directory. You will still need to supply a kmer size to indicate the amount of overlap between contigs.
`-F`	`--input-list-454=FILE`	File containing a list of SDF directories (1 per line) containing 454 sequences to assemble.
`-I`	`--input-list-file=FILE`	File containing a list of SDF directories (1 per line) containing Illumina sequences to assemble.
`-J`	`--input-list-mate-pair=FILE`	File containing a list of SDF directories (1 per line) containing mate pair sequences to assemble.
`-j`	`--mate-pair`	SDF containing mate pair reads. May be specified 0 or more times.
`-o`	`--output=DIR`	Specifies the directory where results are reported.
	`SDF+`	SDF directories containing Illumina sequences to assemble. May be specified 0 or more times.

Sensitivity Tuning		
	`--consensus-reads=INT`	When using read mappings to disambiguate a graph, paths that are supported by fewer reads than the threshold supplied here will not be collapsed (Default is 0).
`-k`	`--kmer-size=INT`	K-mer length to use when constructing a de Bruijn graph.
`-M`	`--max-insert=INT`	Maximum insert size between fragments. (Default is automatically calculated.)
`-m`	`--min-insert=INT`	Minimum insert size between fragments. (Default is automatically calculated.)
`-p`	`--min-path=INT`	Prior to generating a consensus, long paths will be deleted if they are supported by fewer than `--min-path` reads.
`-c`	`--minimum-kmer-frequency=INT`	Set minimum k-mer frequency to retain, or -1 for automatic threshold (Default is -1).
`-a`	`--mismatches=INT`	Number of bases that may mismatch in an alignment or percentage of read that may mismatch (Default is 0).
	`--preserve-bubbles=FLOAT`	Avoid merging bubbles where the ratio of k-mers on the branches is below this (Default is 0.0). This can be used if you wish to preserve diploid information or some near repeats in graph construction.
`-r`	`--read-count=INT`	Prior to generating a consensus delete links in the graph that are supported by fewer reads than this threshold.
`-s`	`--step=INT`	Step size for mapping (Default is 18).
`-w`	`--word=INT`	Word size for mapping (Default is 18).

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
`-T`	`--threads=INT`	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

The `assemble` command attempts to construct long contigs from a large number of short reads. The reads must be converted into SDFs prior to assembly. Illumina reads can be supplied with either the unnamed flag or the `-I` flag, while 454 reads are supplied with `-f` or `-F`. This lets the assembler know the orientation of pairs and which alignment strategy to use. Alternatively this command can be used to improve an existing graph, by mapping additional reads or applying additional filters.

Output

The output of this command is a number of directories in the RTG assembly graph format (documented separately) at each stage of the assembly. The consensus assembly is in the 'final' directory.

```
assemble.log     - log file for the run
build/collapsed  - contigs after tip removal/before bubble popping
build/contigs    - graph prior to tip removal
build/popped     - graph after bubble popping
done             - file that is created when run completes
final            - final consensus graph
mapped           - graph including read mapping paths and counts
progress         - progress file for the run
unfiltered_final - consensus graph which preserves information about merged nodes
```

Graph Construction

The first stage is the construction of a de Bruijn graph and the initial contig construction this includes tip removal and bubble merging. This produces the `build/popped` output directory. This stage may be skipped by using `--graph` to supply an existing graph. The `--minimum-kmer-frequency` (`-c`) flag affects the number of

hashes that will be interpreted as being due to read error, and will be discarded when generating contigs. If -1 is used the first local minimum in the hash frequency distribution will be automatically selected.

Read Mapping

The second stage is to map and pair the original reads against the contig graph. For each read/pair alignments we attempt to find a unique alignment at the best score within the graph. Alignments may cross multiple contigs. If a read/pair maps entirely within a single contig then that contig will have it's 'readCount' attribute incremented. Reads/pairs that map along a series of contigs will increment the 'readCount' of a path joining those contigs.

If you would like to manually specify the insert sizes rather than rely on the automatically calculated fragment sizes you can use the `--max-insert` (`-M`) and `--min-insert` (`-m`) flags. Insert size is measured as the number of bases in between the reads (from the end of the first alignment, to the start of the second). An insert size of -10 indicates that the two fragments overlap by 10 bases, while 20 would mean that there is a gap of 20 bases between alignments. If `-m` and `-M` are omitted read mating distributions will be estimated using the distance between read pairs that are mapped within a single contig, if initial graph construction results in a highly fragmented graph or the insert size is large there may not be enough pairs mapping within a single contig to give an accurate estimate.

Filtering

At this point optional filtering of mappings/paths can occur. You can use the `--min-path` (`-p`) flag to discard paths that are not supported by a significant number of reads. The `--read-count` (`-r`) flag will disconnect links with low mapping counts. The best value for `--read-count` should be related to the coverage of the sample. Higher values can often result in longer contigs but may result in a more fragmented assembly graph.

Consensus

Finally read mappings are used to resolve ambiguities and repeats in the contig graph. The result is written to the `final` directory. Within consensus generation paths containing more than `--consensus-reads` mapped will potentially be merged into a single contig. Increasing this may help to reduce mis-assemblies.

See also:

format

2.6.2 addpacbio

Synopsis:

The `addpacbio` command adds long reads to an existing assembly to enable an improved consensus.

Syntax:

Map a set of pacbio reads to an assembled graph:

```
$ rtg addpacbio -g DIR -o DIR SDF+
```

Example:

```
$ rtg addpacbio --trim -g initial_assembly/final -o output pac_bio.sdf
```

Parameters:

File Input/Output		
-g	--graph=DIR	Graph of the assembly to map against.
-I	--input-list-file=FILE	File containing a list of SDF directories (1 per line) containing sequences to assemble.
-o	--output=DIR	Directory for output.
	--trim	Before mapping remove any short disconnected sequences from the graph.
	SDF+	SDF directories containing reads to map. May be specified 0 or more times.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

The `addpacbio` command uses an alternate mapping scheme designed to handle Pacific Biosciences reads which are longer with a higher error rate. The reads must be converted into SDF format prior to mapping. The input graph will usually have been constructed from short reads.

The `--trim` option causes short contigs (<200 bp) that don't add connections to the graph to be removed. These sequences don't contribute to the consensus and are often highly repetitive resulting in lots of work for the mapper. Setting this option will often result in much faster execution.

See also:

assemble, *format*

2.7 Variant Detection Commands

2.7.1 snp

Synopsis:

The `snp` command calls sequence variants, such as single nucleotide polymorphisms (SNPs), multi-nucleotide polymorphisms (MNPs) and indels, from a set of alignments reported in genome-position sorted SAM/BAM. Bayesian statistics are used to determine the likelihood that a given base pair will be a SNP (either homozygous or heterozygous) given the sample evidence represented in the read alignments and prior knowledge about the experiment.

Syntax:

Multi-file input specified from command line:

```
$ rtg snp [OPTION]... -o DIR -t SDF FILE+
```

Multi-file input specified in a text file:

```
$ rtg snp [OPTION]... -o DIR -t SDF -I FILE
```

Example:

```
$ rtg snp -o hs_snp -t hs_reference hs_map/alignments.bam
```

Parameters:

File Input/Output		
	`--bed-regions=FILE`	If set, only read SAM records that overlap the ranges contained in the specified BED file.
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-o`	`--output=DIR`	Directory for output.
	`--region=REGION`	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	--enable-allelic-fraction	If set, incorporate the expected allelic fraction in scoring.
	--exclude-mated	Exclude all mated SAM records.
	--exclude-unmated	Exclude all unmated SAM records.
	--keep-duplicates	Don't detect and filter duplicate reads based on mapping position.
-m	--machine-errors=STRING	If set, force sequencer machine settings. Allowed values are [default, illumina, ls454_se, ls454_pe, complete, iontorrent]
	--max-as-mated=INT	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
	--max-as-unmated=INT	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
	--max-coverage=INT	Skip calling in sites with per sample read depth exceeding this value (Default is 200)
	--max-coverage-multiplier=FLOAT	Skip calling in sites with combined depth exceeding multiplier * average combined coverage determined from calibration (Default is 5.0)
	--max-hits=INT	If set, ignore SAM records with an alignment count that exceeds this value.
	--min-base-quality=INT	Phred scaled quality score, read bases below this quality will be treated as unknowns (Default is 0)
	--min-mapq=INT	If set, ignore SAM records with MAPQ less than this value.
	--min-variant-allelic-depth=FLOAT	If set, also output sites that meet this minimum quality-adjusted alternate allelic depth.
	--min-variant-allelic-fraction=FLOAT	If set, also output sites that meet this minimum quality-adjusted alternate allelic fraction.
-p	--pedigree=FILE	Genome relationships PED file containing sex of individual.
	--ploidy=STRING	Ploidy to use. Allowed values are [auto, diploid, haploid] (Default is auto)
	--population-priors=FILE	If set, use the VCF file to generate population based site-specific priors.
	--rdefault-mated=INT	For mated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	--rdefault-unmated=INT	For unmated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	--sex=SEX	Sex of individual. Allowed values are [male, female, either] (Default is either)

Reporting		
-a	--all	Write variant calls covering every position irrespective of thresholds.
	--avr-model=MODEL	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-wgs.avr)
	--filter-ambiguity=INT	Threshold for ambiguity filter applied to output variants.
	--filter-bed=FILE	Apply a position based filter, retaining only variants that fall in these BED regions.
	--filter-depth=INT	Apply a fixed depth of coverage filter to output variants.
	--filter-depth-multiplier=FLOAT	Apply a ratio based depth filter. The filter will be multiplier * average coverage determined from calibration files.
	--min-avr-score=FLOAT	If set, fail variants with AVR scores below this value.
	--snps-only	If set, will output simple SNPs only.

Utility		
-h	--help	Print help on command-line flag usage.
	--no-calibration	If set, ignore mapping calibration files.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

During variant calling, a posterior distribution is calculated for each variant, which represents the knowledge gained from the combination of prior estimates given the nature of the experiment and the actual evidence of the read alignments. The mean of the posterior distribution is calculated and displayed with the results.

The default Bayesian model does not directly include any expectation of allelic fraction, however, for typical germline calling it is expected that heterozygous variants should have approximately equal support both alleles at a site. The `--enable-allelic-fraction` instructs the variant caller to include a term in the model which accounts for the expected allelic fraction. This parameter reduces incorrect variant calls overall, but there may be a loss of sensitivity to variants which do not follow normal germline expectations, such as mosaic *de novo* variants. The user should decide whether to enable this flag according to their needs.

The output of the `snp` command is industry standard VCF that includes each variant called with confidence. The location and type of the call, the base pairs (reference and called), and a confidence score are standard output. Additional support statistics describe read alignment evidence that can be used to evaluate confidence in the called SNPs.

The `--all` flag produces calls at all non-N base positions in the reference irrespective of thresholds and whether a variant is called at each position. Some calls cover multiple positions so there may not be a separate call for every nucleotide. This can be very useful for creating a full-reference consensus or for summarizing pileup information in a text format file. However, the resulting output is quite large (one output line per base pair in the reference), which takes longer to process and requires considerably more space to store.

Note: For more information about the `snps.vcf` output file column definitions, see *Small-variant VCF output file description*.

Quality calibration

Read data from Complete Genomics and from manufacturers that supply data in FASTQ format include a quality value for each base of the reads. This indicates the probability of an error in the base calling at that position

in the read. Following industry best practice we calculate recalibration tables using data from the mapping process. These calibration files are automatically generated in the `map` and `cgmap` commands or can be manually generated using the `calibrate` command. The `snp` command automatically detects the calibration files using the mapping file locations. To run variant calling without calibration information for the alignment files, set the `--no-calibration` flag. Note that without calibration information, the variant calling will have no knowledge about the expected sequencing coverage levels, so you should set an appropriate `--max-coverage` value. Failure to set an appropriate value may result in under-calling, particularly for complex variants such as indels.

Coverage filtering

The variant calls made in regions of excessive coverage are often due to incorrect mappings, particularly with short reads. The `snp` command allows you to apply a maximum coverage filter with the `--filter-depth` and `--filter-depth-multiplier` parameters.

Similarly, regions of excessive coverage can negatively impact variant calling speed so a separate set of flags allow calling to be skipped in regions of excessive coverage. These regions are noted in the `regions.bed` file as an extreme coverage region. Under normal circumstances, calibration information is used to automatically select a coverage threshold – the maximum coverage cutoff is calculated by multiplying a coverage multiplier with the average coverage for the genome sequence (the default multiplier is 5.0). The `--max-coverage-multiplier` parameter can be used to adjust the multiplier. When recalibration information is not available, the maximum coverage cutoff is determined using the `--max-coverage` parameter, which sets a fixed value as the threshold (the default is 200).

Prior distributions

The use of a prior distribution can increase the likelihood of calling novel variants by increasing the confidence that sample evidence supports a particular variant hypothesis. With priors, the calculated range of likely variants is smaller than that expected with a normal distribution. Currently, the genome-wide prior distribution is set by default for the human genome (for adjusted genome priors, contact Real Time Genomics technical support for assistance). An alternative is to supply site-specific prior information in the form of a VCF containing variants with allele-frequency information via the `--population-priors` flag. This will adjust the likelihood of calling variants that have been seen before in the population.

Adaptive Variant Rescoring

The RTG Adaptive Variant Rescoring (AVR) system uses machine learning to build adaptive models that take into account factors not already accounted for in the Bayesian statistics model in determining the probability that a given variant call is correct. Some pre-built AVR models are provided with the RTG software, to build your own models you can use the `avrbuild` command with VCF output from RTG variant callers filtered to a set of known correct calls and a set of known incorrect calls. These models when used either directly by the variant callers, or when applied using the `avrpredict` command produce a VCF format field called AVR which contains a probability between 0 and 1 that the call is correct. This can then be used to filter your results to remove calls unlikely to be correct.

See also:

vcffilter, *vcfannotate*, *coverage*, *cnv*, *family*, *somatic*, *population*, *calibrate*

2.7.2 family

Synopsis:

The `family` command calls sequence variants on a combination of individuals using Mendelian inheritance.

Syntax:

Relationships specified via pedigree file, with multi-file input specified from command line:

```
$ rtg family [OPTION]... -o DIR -t SDF -p FILE FILE+
```

Relationships specified via pedigree file, with multi-file input specified in a text file:

```
$ rtg family [OPTION]... -o DIR -t SDF -p FILE -I FILE
```

Relationships specified via flags, with multi-file input specified from command line:

```
$ rtg family [OPTION]... -o DIR -t SDF --father STRING --mother STRING \
  <--daughter STRING | --son STRING>+ FILE+
```

Relationships specified via flags, with multi-file input specified in a text file:

```
$ rtg family [OPTION]... -o DIR -t SDF --father STRING --mother STRING \
  <--daughter STRING | --son STRING>+ -I FILE
```

Example:

```
$ rtg family -o fam -t reference --father f_sample --mother m_sample \
  --daughter d_sample --son s_sample -I samfiles.txt
```

Parameters:

File Input/Output		
	--bed-regions=FILE	If set, only read SAM records that overlap the ranges contained in the specified BED file.
	--daughter=STRING	Sample identifier used in read groups for a daughter sample. May be specified 0 or more times.
	--father=STRING	Sample identifier used in read groups for father sample.
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
	--mother=STRING	Sample identifier used in read groups for for mother sample.
-o	--output=DIR	Directory for output.
-p	--pedigree=FILE	Genome relationships PED file, if not specifying family via –father, –mother, etc.
	--region=REGION	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
	--son=STRING	Sample identifier used in read groups for a son sample. May be specified 0 or more times.
-t	--template=SDF	SDF containing the reference genome.
	FILE+	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	`--enable-allelic-fraction`	If set, incorporate the expected allelic fraction in scoring.
	`--keep-duplicates`	Don't detect and filter duplicate reads based on mapping position.
`-m`	`--machine-errors=STRING`	If set, force sequencer machine settings. Allowed values are [default, illumina, ls454_se, ls454_pe, complete, iontorrent]
	`--max-coverage=INT`	Skip calling in sites with per sample read depth exceeding this value (Default is 200)
	`--max-coverage-multiplier=FLOAT`	Skip calling in sites with combined depth exceeding multiplier * average combined coverage determined from calibration (Default is 5.0)
	`--min-base-quality=INT`	Phred scaled quality score, read bases below this quality will be treated as unknowns (Default is 0)
	`--min-mapq=INT`	If set, ignore SAM records with MAPQ less than this value.
	`--population-priors=FILE`	If set, use the VCF file to generate population based site-specific priors.
	`--rdefault-mated=INT`	For mated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	`--rdefault-unmated=INT`	For unmated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)

Reporting		
`-a`	`--all`	Write variant calls covering every position irrespective of thresholds.
	`--avr-model=MODEL`	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-wgs.avr)
	`--filter-ambiguity=INT`	Threshold for ambiguity filter applied to output variants.
	`--filter-bed=FILE`	Apply a position based filter, retaining only variants that fall in these BED regions.
	`--filter-depth=INT`	Apply a fixed depth of coverage filter to output variants.
	`--filter-depth-multiplier=FLOAT`	Apply a ratio based depth filter. The filter will be multiplier * average coverage determined from calibration files.
	`--min-avr-score=FLOAT`	If set, fail variants with AVR scores below this value.
	`--snps-only`	If set, will output simple SNPs only.

Utility		
`-h`	`--help`	Print help on command-line flag usage.
	`--no-calibration`	If set, ignore mapping calibration files.
`-Z`	`--no-gzip`	Do not gzip the output.
`-T`	`--threads=INT`	Number of threads (Default is the number of available cores)

Usage:

The `family` command jointly calls samples corresponding to the parents and children of a family using Mendelian inheritance. The `family` command requires a sample for each of the father, mother one or more children, either daughters or sons.

The family relationships and sample sexes can be supplied either by the use of separate flags that indicate the sex and relationship of each sample within the family, or by supplying a pedigree file containing this information. See

ref:*Pedigree PED input file format*.

The `family` command works by considering all the evidence at each nucleotide position and makes a joint Bayesian estimate that a given nucleotide position represents a variant in one or more of the samples. As with the `snp` command, some calls may extend across multiple adjacent nucleotide positions.

The `family` command requires that each sample has appropriate read group information specified in the BAM files created during mapping. For information about how to specify read group information when mapping see *Using SAM/BAM Read Groups in RTG map*.

By default the VCF output consists of calls where one or more samples differ from the reference genome. The `--all` flag produces calls at all non-N base positions for which there is some evidence, irrespective of thresholds and whether or not the call is equal to the reference. Using `--all` can incur a significant performance penalty and is best applied only in small regions of interest (selected with the `--region` or `--bed-regions` options).

When there is sufficient evidence, a call may be made that violates Mendelian inheritance consistency. When this happens the output VCF will contain a `DN` format field which will indicate if the call for a given sample is presumed to be a *de novo* call. This will also be accompanied by a `DNP` format field which contains a Phred scaled probability that the call is due to an actual *de novo* variant.

When a child can be unambiguously phased according to Mendelian inheritance, the VCF genotype field (`GT`) will use the phased separator `|` instead of the unphased separator `/`. The genotype field will be ordered such that the allele inherited from the father is first, and that from the mother is second.

For details concerning quality calibration, prior distributions and adaptive variant rescoring refer to the information for the `snp` command in *snp*.

See also:

snp, *somatic*, *population*, *calibrate*

2.7.3 somatic

Synopsis:

The `somatic` command calls sequence variants on an original and derived sample set.

Syntax:

Multi-file input specified from command line:

```
$ rtg somatic [OPTION]... -o DIR -t SDF --contamination FLOAT --derived STRING \
  --original STRING FILE+
```

Multi-file input specified in a text file:

```
$ rtg somatic [OPTION]... -o DIR -t SDF --contamination FLOAT --derived STRING \
  --original STRING -I FILE
```

Example:

```
$ rtg somatic -o som -t reference --derived c_sample --original n_sample \
  --contamination 0.3 -I samfiles.txt
```

Parameters:

File Input/Output		
	`--bed-regions=FILE`	If set, only read SAM records that overlap the ranges contained in the specified BED file.
	`--derived=STRING`	Sample identifier used in read groups for derived sample.
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
	`--original=STRING`	Sample identifier used in read groups for original sample.
`-o`	`--output=DIR`	Directory for output.
	`--region=REGION`	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	--contamination=FLOAT	Estimated fraction of contamination in derived sample.
	--enable-allelic-fraction	If set, incorporate the expected allelic fraction in scoring.
	--enable-somatic-allelic-fraction	If set, incorporate the expected somatic allelic fraction in scoring.
-G	--include-gain-of-reference	Include gain of reference somatic calls in output VCF.
	--include-germline	Include germline variants in output VCF.
	--keep-duplicates	Don't detect and filter duplicate reads based on mapping position.
	--loh=FLOAT	Prior probability that a loss of heterozygosity event has occurred (Default is 0.0)
-m	--machine-errors=STRING	If set, force sequencer machine settings. Allowed values are [default, illumina, ls454_se, ls454_pe, complete, iontorrent]
	--max-coverage=INT	Skip calling in sites with per sample read depth exceeding this value (Default is 200)
	--max-coverage-multiplier=FLOAT	Skip calling in sites with combined depth exceeding multiplier * average combined coverage determined from calibration (Default is 25.0)
	--min-base-quality=INT	Phred scaled quality score, read bases below this quality will be treated as unknowns (Default is 0)
	--min-mapq=INT	If set, ignore SAM records with MAPQ less than this value.
	--min-variant-allelic-depth=FLOAT	If set, also output sites that meet this minimum quality-adjusted alternate allelic depth.
	--min-variant-allelic-fraction=FLOAT	If set, also output sites that meet this minimum quality-adjusted alternate allelic fraction.
	--population-priors=FILE	If set, use the VCF file to generate population based site-specific priors.
	--rdefault-mated=INT	For mated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	--rdefault-unmated=INT	For unmated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	--sex=SEX	Sex of individual. Allowed values are [male, female, either] (Default is either)
-s	--somatic=FLOAT	Default prior probability of a somatic SNP mutation in the derived sample (Default is 0.000001)
	--somatic-priors=FILE	If set, use the BED file to generate site specific somatic priors.

Reporting		
-a	--all	Write variant calls covering every position irrespective of thresholds.
	--avr-model=MODEL	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-somatic.avr)
	--filter-ambiguity=INT	Threshold for ambiguity filter applied to output variants.
	--filter-bed=FILE	Apply a position based filter, retaining only variants that fall in these BED regions.
	--filter-depth=INT	Apply a fixed depth of coverage filter to output variants.
	--filter-depth-multiplier=FLOAT	Apply a ratio based depth filter. The filter will be multiplier * average coverage determined from calibration files.
	--min-avr-score=FLOAT	If set, fail variants with AVR scores below this value.
	--snps-only	If set, will output simple SNPs only.

Utility		
-h	--help	Print help on command-line flag usage.
	--no-calibration	If set, ignore mapping calibration files.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

The `somatic` command performs a joint calling on an original sample corresponding to ordinary cells and a derived sample corresponding to cancerous cells. The derived sample may be contaminated with the original sample and the contamination level should be specified. It is also desirable that a prior probability of somatic mutation be set. To compute a rough estimate for this, make an estimate of the number of mutations expected and divide it by the length of the genome.

The `somatic` command works by considering all the evidence at each nucleotide position and makes a joint Bayesian estimate that a given nucleotide position represents a somatic mutation in the derived sample. As with the `snp` command, some calls may extend across multiple adjacent nucleotide positions.

The `somatic` command requires that each sample has appropriate read group information specified in the BAM files created during mapping. For information about how to specify read group information when mapping see *Using SAM/BAM Read Groups in RTG map*.

By default the VCF output consists of calls for both samples where there is a difference between the original and derived sample. The `--all` flag produces calls at all non-N base positions for which there is some evidence, irrespective of thresholds and whether or not the call is equal to the reference. Using `--all` can incur a significant performance penalty and is best applied only in small regions of interest (selected with the `--region` or `--bed-regions` options). More information regarding VCF fields output by the somatic command is given in *Small-variant VCF output file description*.

As with the default germline calling, the Bayesian model employed during somatic calling does not directly include any expectation of allelic fraction for somatic variants, however, for tumors with low heterogeneity it is expected that somatic variants should appear with particular allelic fraction according to the contamination level. The `--enable-somatic-allelic-fraction` instructs the variant caller to include a term in the model which accounts for the expected allelic fraction of somatic variants. This parameter reduces incorrect somatic calls overall, but may not be appropriate if the tumor heterogeneity is high or if contamination is not well known. The user should decide whether to enable this flag according to their needs. This flag can be used in conjunction with `--enable-allelic-fraction`.

By default `somatic` scores variants with a model trained on somatic variants. If you are interested in the germline calls (of either the normal or tumor sample), then it is preferable to use a different AVR model, for example by adding `--avr-model illumina-wgs.avr` to the command line. Alternatively, using `avrpredict`, it is possible after the run has completed to rescore according to a different model.

The `--loh` parameter is used to control the sensitivity to variants occurring in regions of loss of heterozygosity. In heterozygous regions, a somatic mutation of the form $XY \rightarrow ZZ$ (with $X \neq Z$ and $Y \neq Z$) is extremely unlikely; however, in a loss of heterozygosity region, $XY \rightarrow Z$ is plausible. As the loss of heterozygosity prior is increased, the barrier to detecting and reporting such variants is reduced. If a region is known or suspected to have a loss of heterozygosity, then a value close to 1 should be used when calling that region.

The `--somatic-priors` option allows fine-grained control over the prior probability of a site being somatic. For further detail see *Using site-specific somatic priors*.

For details concerning quality calibration prior distributions refer to the information for the `snp` command in *snp*.

See also:

snp, *family*, *population*, *calibrate*, *avrpredict*

2.7.4 population

Synopsis:

The `population` command calls sequence variants on a set of individuals.

Syntax:

Multi-file input specified from command line:

```
$ rtg population [OPTION]... -o DIR -p FILE -t SDF FILE+
```

Multi-file input specified in a text file:

```
$ rtg population [OPTION]... -o DIR -p FILE -t SDF -I FILE
```

Example:

```
$ rtg population -o pop -p relations.ped -t reference -I samfiles.txt
```

Parameters:

File Input/Output		
	`--bed-regions=FILE`	If set, only read SAM records that overlap the ranges contained in the specified BED file.
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-o`	`--output=DIR`	Directory for output.
`-p`	`--pedigree=FILE`	Genome relationships PED file.
	`--region=REGION`	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	--enable-allelic-fraction	If set, incorporate the expected allelic fraction in scoring.
	--keep-duplicates	Don't detect and filter duplicate reads based on mapping position.
-m	--machine-errors=STRING	If set, force sequencer machine settings. Allowed values are [default, illumina, ls454_se, ls454_pe, complete, iontorrent]
	--max-coverage=INT	Skip calling in sites with per sample read depth exceeding this value (Default is 200)
	--max-coverage-multiplier=FLOAT	Skip calling in sites with combined depth exceeding multiplier * average combined coverage determined from calibration (Default is 5.0)
	--min-base-quality=INT	Phred scaled quality score, read bases below this quality will be treated as unknowns (Default is 0)
	--min-mapq=INT	If set, ignore SAM records with MAPQ less than this value.
	--pedigree-connectivity=STRING	Sets mode of operation based on how well connected the pedigree is. Allowed values are [auto, sparse, dense] (Default is auto)
	--population-priors=FILE	If set, use the VCF file to generate population based site-specific priors.
	--rdefault-mated=INT	For mated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	--rdefault-unmated=INT	For unmated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)

Reporting		
-a	--all	Write variant calls covering every position irrespective of thresholds.
	--avr-model=MODEL	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-wgs.avr)
	--filter-ambiguity=INT	Threshold for ambiguity filter applied to output variants.
	--filter-bed=FILE	Apply a position based filter, retaining only variants that fall in these BED regions.
	--filter-depth=INT	Apply a fixed depth of coverage filter to output variants.
	--filter-depth-multiplier=FLOAT	Apply a ratio based depth filter. The filter will be multiplier * average coverage determined from calibration files.
	--impute=STRING	Name of sample absent from mappings to impute genotype for. May be specified 0 or more times, or as a comma separated list.
	--min-avr-score=FLOAT	If set, fail variants with AVR scores below this value.
	--snps-only	If set, will output simple SNPs only.

Utility		
-h	--help	Print help on command-line flag usage.
	--no-calibration	If set, ignore mapping calibration files.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

The `population` command performs a joint calling on a set of samples corresponding to multiple individuals from a population.

The `population` command works by considering all the evidence at each nucleotide position and makes a joint Bayesian estimate that a given nucleotide position represents a variant in one or more of the samples. As with the `snp` command, some calls may extend across multiple adjacent nucleotide positions.

The `population` command requires that each sample has appropriate read group information specified in the BAM files created during mapping. For information about how to specify read group information when mapping see *Using SAM/BAM Read Groups in RTG map*. Also required is a pedigree file describing the samples being processed, so that the caller can utilize pedigree information to improve the variant calling accuracy. This is provided in a PED format file using the `--pedigree` flag. For more information about the PED file format see *Pedigree PED input file format*.

The `--pedigree-connectivity` flag allows the specification of different modes for the population caller to run in based on how well connected the pedigree samples are.

The `dense` pedigree mode assumes that there are one or more samples connected by a pedigree. This can in principle be used for a single sample or for a family specified in the pedigree. It can also process a pedigree where there are many disconnected samples or fragments of pedigrees. However, it may be more appropriate to use the `sparse` mode in this case.

The `sparse` pedigree mode is intended for the case where there are many separate samples with no directly known pedigree connections. It uses Hardy-Weinberg equilibrium to ensure that the calls in the different samples are consistent with one another. Doing this may take more time than for the `dense` pedigree mode but will give better results when the samples are not connected by a pedigree. It is also useful when the pedigree consists of a large number of separate families or more complex situations where there are mixed separate samples and families or larger fragments of pedigrees.

The default `auto` setting selects `dense` pedigree mode when the called samples form fewer than three disconnected pedigree fragments, otherwise `sparse` mode is used.

The decision about whether to use the `dense` or `sparse` pedigree mode is not necessarily clear cut. If you have tens of separate families or samples then using the `sparse` pedigree mode will definitely improve performance (at the expense of additional run time). If you have only one or two families or samples or a single large connected pedigree then using the `dense` pedigree mode will be the best solution. When the coverage is lower the `sparse` pedigree mode will be more valuable. When significant prior information is available in the form of a prior VCF file, then the `sparse` mode will be less valuable.

By default the VCF output consists of calls where one or more samples differ from the reference genome. The `--all` flag produces calls at all non-N base positions for which there is some evidence, irrespective of thresholds and whether or not the call is equal to the reference. Using `--all` can incur a significant performance penalty and is best applied only in small regions of interest (selected with the `--region` or `--bed-regions` options).

When there is sufficient evidence, a call may be made that violates Mendelian inheritance consistency for family groupings in the pedigree. When this happens the output VCF will contain a `DN` format field which will indicate if the call for a given sample is presumed to be a *de novo* call. This will also be accompanied by a `DNP` format field which contains a Phred scaled probability that the call is due to an actual *de novo* variant.

When a variant call on a child in the pedigree can be unambiguously phased according to Mendelian inheritance, the VCF genotype field (GT) will use the phased separator `|` instead of the unphased separator `/`. The genotype field will be ordered such that the allele inherited from the father is first, and the mothers is second.

For details concerning quality calibration, prior distributions and adaptive variant rescoring refer to the information for the `snp` command in *snp*.

See also:

snp, *family*, *somatic*, *calibrate*

2.7.5 lineage

Synopsis:

The `lineage` command calls sequence variants on a set of cell lineage samples.

Syntax:

Multi-file input specified from command line:

```
$ rtg lineage [OPTION]... -o DIR -p FILE -t SDF FILE+
```

Multi-file input specified in a text file:

```
$ rtg lineage [OPTION]... -o DIR -p FILE -t SDF -I FILE
```

Example:

```
$ rtg lineage -o lin -p relations.ped -t reference -I samfiles.txt
```

Parameters:

File Input/Output		
	`--bed-regions=FILE`	If set, only read SAM records that overlap the ranges contained in the specified BED file.
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-o`	`--output=DIR`	Directory for output.
`-p`	`--pedigree=FILE`	Genome relationships PED file.
	`--region=REGION`	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	`--enable-allelic-fraction`	If set, incorporate the expected allelic fraction in scoring.
	`--keep-duplicates`	Don't detect and filter duplicate reads based on mapping position.
`-m`	`--machine-errors=STRING`	If set, force sequencer machine settings. Allowed values are [default, illumina, ls454_se, ls454_pe, complete, iontorrent]
	`--max-coverage=INT`	Skip calling in sites with per sample read depth exceeding this value (Default is 200)
	`--max-coverage-multiplier=FLOAT`	Skip calling in sites with combined depth exceeding multiplier * average combined coverage determined from calibration (Default is 5.0)
	`--min-base-quality=INT`	Phred scaled quality score, read bases below this quality will be treated as unknowns (Default is 0)
	`--min-mapq=INT`	If set, ignore SAM records with MAPQ less than this value.
	`--population-priors=FILE`	If set, use the VCF file to generate population based site-specific priors.
	`--rdefault-mated=INT`	For mated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)
	`--rdefault-unmated=INT`	For unmated reads that have no mapping quality supplied use this as the default quality (in Phred format from 0 to 63) (Default is 20)

Reporting		
-a	--all	Write variant calls covering every position irrespective of thresholds.
	--avr-model=MODEL	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-wgs.avr)
	--filter-ambiguity=INT	Threshold for ambiguity filter applied to output variants.
	--filter-bed=FILE	Apply a position based filter, retaining only variants that fall in these BED regions.
	--filter-depth=INT	Apply a fixed depth of coverage filter to output variants.
	--filter-depth-multiplier=FLOAT	Apply a ratio based depth filter. The filter will be multiplier * average coverage determined from calibration files.
	--min-avr-score=FLOAT	If set, fail variants with AVR scores below this value.
	--snps-only	If set, will output simple SNPs only.

Utility		
-h	--help	Print help on command-line flag usage.
	--no-calibration	If set, ignore mapping calibration files.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

The `lineage` command performs a joint calling on a set of samples from a cell lineage.

The `lineage` command works by considering all the evidence at each nucleotide position and makes a joint Bayesian estimate that a given nucleotide position represents a variant in one or more of the samples. As with the `snp` command, some calls may extend across multiple adjacent nucleotide positions.

The `lineage` command requires that each sample has appropriate read group information specified in the BAM files created during mapping. For information about how to specify read group information when mapping see *Using SAM/BAM Read Groups in RTG map*. Also required is a pedigree file describing the samples being processed, so that the caller can utilize pedigree information to improve the variant calling accuracy. This is provided in a PED format file using the `--pedigree` flag. For more information about the PED file format see *Pedigree PED input file format*.

By default the VCF output consists of calls where one or more samples differ from the reference genome. The `--all` flag produces calls at all non-N base positions for which there is some evidence, irrespective of thresholds and whether or not the call is equal to the reference. Using `--all` can incur a significant performance penalty and is best applied only in small regions of interest (selected with the `--region` or `--bed-regions` options).

For details concerning quality calibration, prior distributions and adaptive variant rescoring refer to the information for the `snp` command in *snp*.

See also:

snp, *family*, *somatic*, *population*, *calibrate*

2.7.6 avrpredict

Synopsis:

The `avrpredict` command is used to score variants in a VCF file using an adaptive variant rescoring model.

Syntax:

```
$ rtg avrpredict [OPTION]... -i FILE -o FILE
```

Example:

```
$ rtg avrpredict -i snps.vcf.gz --avr-model avr.model -o avr.vcf.gz
```

Parameters:

File Input/Output		
-i	--input=FILE	Input VCF file containing variants to score. Use '-' to read from standard input.
-o	--output=FILE	Output VCF file. Use '-' to write to standard output.

Reporting		
	--avr-model=MODEL	Name of AVR model to use when scoring variants. Allowed values are [alternate.avr, illumina-exome.avr, illumina-somatic.avr, illumina-wgs.avr, none] or a path to a model file (Default is illumina-wgs.avr)
	--min-avr-score=FLOAT	If set, fail variants with AVR scores below this value.
-s	--sample=STRING	If set, only re-score the specified samples (Default is to re-score all samples). May be specified 0 or more times.
-f	--vcf-score-field=STRING	The name of the VCF FORMAT field in which to store the computed score (Default is AVR)

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.

Usage:

Used to apply an adaptive variant rescoring model to an existing VCF file produced by an RTG variant caller. The output VCF will contain a new or updated AVR score field for the samples to which the model is being applied. This can be used in combination with the `avrbuild` command to produce AVR scores from more detailed training data for a given experiment.

By default `avrpredict` will write the score into the AVR field of the specified sample in the VCF. However, it is possible to specify a different score field name using `--vcf-score-field` and this can be useful when there are multiple applicable AVR models (e.g. scoring using both somatic and germline models).

See also:

avrbuild, *avrstats*, *snp*, *family*, *population*

2.7.7 avrbuild

Synopsis:

The `avrbuild` command is used to create adaptive variant rescoring models from positive and negative training examples.

Syntax:

```
$ rtg avrbuild [OPTION]... -n FILE -o FILE -p FILE
```

Example:

```
$ rtg avrbuild -o avr.model -n fp.vcf.gz -p tp.vcf.gz --format-annotations GQ,DP
```

Parameters:

File Input/Output		
-a	--annotated=FILE	VCF file containing training examples annotated with CALL=TP/FP. May be specified 0 or more times.
	--bed-regions=FILE	If set, only read VCF records that overlap the ranges contained in the specified BED file.
-n	--negative=FILE	VCF file containing negative training examples. May be specified 0 or more times.
-o	--output=FILE	Output AVR model.
-p	--positive=FILE	VCF file containing positive training examples. May be specified 0 or more times.

Sensitivity Tuning		
	--derived-annotations=STRING	Derived fields to use in model. Allowed values are [IC, EP, LAL, QD, NAA, AN, GQD, VAF1, ZY, PD, MEANQAD, QA, RA]. May be specified 0 or more times, or as a comma separated list.
	--format-annotations=STRING	FORMAT fields to use in model. May be specified 0 or more times, or as a comma separated list.
	--info-annotations=STRING	INFO fields to use in model. May be specified 0 or more times, or as a comma separated list.
	--qual-annotation	If set, use QUAL annotation in model.
-s	--sample=STRING	The name of the sample to select (required when using multi-sample VCF files)

Utility		
-h	--help	Print help on command-line flag usage.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

Used to produce an adaptive variant rescoring model using machine learning on a set of variants produced by RTG that have been divided into known positive and negative examples. The model will learn how to work out how likely a call is correct based on the set of annotations provided on the command line extracted from the input VCF files.

Input training VCF files are typically supplied as separate sets of positive and negative training examples via `--positive` and `--negative` options.

An alternative is to supply VCF files where training instances have been annotated with their training status, using the `--annotated` option. The annotation format is the same as that produced by `vcfeval` when using `--output-mode=annotate`, so you can supply the `calls.vcf.gz` file produced by such runs directly to `avrbuild`.

The model file produced can then be used directly when variant calling to produce an AVR score field by using the `--avr-model` parameter, or applied to an existing VCF output file using the `avrpredict` command.

For details concerning the various VCF fields available for training see the Appendix *Small-variant VCF output file description*. The derived annotations are those which can either be present in the VCF record or computed from other fields in the VCF record.

See also:

avrpredict, *avrstats*, *snp*, *family*, *population*

2.7.8 svprep

Synopsis:

Prepares mapping output for use with the `sv` and `discord` commands. This functionality is automatically performed by the `map` and `cgmap` commands unless `--no-svprep` was given during mapping, and so does not ordinarily need to be executed separately.

Syntax:

```
$ rtg svprep [OPTION]... DIR
```

Example:

```
$ rtg svprep map_out
```

Parameters:

File Input/Output		
	DIR	Specifies the directory containing SAM/BAM format files for preparation.

Utility		
-h	--help	Print help on command-line flag usage.
	--no-augment	If set, only compute read group statistics.

Usage:

Use the svprep command to prepare mappings for structural variant analysis. The svprep command performs three functions:

- First, it identifies discordant reads (those were there exists a unique unmated mapping for each arm of a paired-end) and fills in the RNEXT/PNEXT/TLEN fields for these records. The augmented unmated SAM/BAM file will replace the original.
- Secondly it identifies unmapped reads for which there exists a unique unmated mapping for the other arm and fills in an estimated position for the unmapped read. The augmented unmapped SAM/BAM file will replace the original.
- Thirdly it generates per read-group statistics on observed length distributions used by subsequent structural variant analysis tools.

svprep may be instructed to perform only the last of these functions via the --no-augment flag.

The svprep functionality is integrated directly into the RTG mapping commands by default, so does not normally need to be executed as a separate stage.

See also:

map, *sv*, *discord*

2.7.9 discord

Synopsis:

Analyses SAM records to determine the location of structural variant break-ends based on discordant mappings.

Syntax:

Multi-file input specified from command line:

```
$ rtg discord [OPTION]... -o DIR -t SDF -r FILE FILE+
```

Multi-file input specified in a text file:

```
$ rtg discord [OPTION]... -o DIR -t SDF -I FILE -R FILE
```

Example:

```
$ rtg discord -o break_out -t genome -I sam-list.txt -R rgstats-list.txt
```

Parameters:

File Input/Output		
	`--bed`	Produce output in BED format in addition to VCF.
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-o`	`--output=DIR`	Directory for output.
`-r`	`--readgroup-stats=FILE`	Text file containing read group stats. May be specified 0 or more times.
`-R`	`--readgroup-stats-list-file=FILE`	File containing list of read group stats files (1 per line)
	`--region=REGION`	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	`--consistent-only`	Only include breakends with internally consistent supporting reads.
`-m`	`--max-as-mated=INT`	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
`-u`	`--max-as-unmated=INT`	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
`-c`	`--max-hits=INT`	If set, ignore SAM records with an alignment count that exceeds this value.
`-s`	`--min-support=INT`	Minimum number of supporting reads for a breakend (Default is 3)
	`--overlap-fraction=FLOAT`	Assume this fraction of an aligned ready may may overlap a breakend (Default is 0.01)

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
`-Z`	`--no-gzip`	Set this flag to create the output files without compression. By default the output files are compressed with tabix compatible blocked gzip.
`-T`	`--threads=INT`	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

This command takes as input a set of mapped and mated reads and a genome. It locates clusters of reads whose mates are not within the expected mating range but clustered somewhere else on the reference, indicating a potential structural variant.

The `discord` command considers each discordantly mapped read and calculates a constraint on the possible locations of the structural variant break-ends. When all discordant reads within a cluster agree on the possible break-end positions, this is considered consistent. It is also possible for the reads within a discordant cluster to be inconsistent, usually this is a spurious call but could indicate a more complex structural variant. By default these break-ends are included in the output VCF but marked as failing a consistency filter.

Also included in the output VCF is an `INFO` field indicating the number of discordant reads contributing to each break-end, which may be useful to filter out spurious calls. Those with too few contributing reads are likely to be incorrect, and similarly those with too many reads are likely to be a result of mapping artifacts.

For additional information about the `discord` command output files see *Discord command output file descriptions*.

See also:

svprep, *sv*

2.7.10 sv

Synopsis:

Analyses SAM records to determine the location of structural variants.

Syntax:

Multi-file input specified from command line:

```
$ rtg sv [OPTION]... -o DIR -t SDF -r FILE FILE+
```

Multi-file input specified in a text file:

```
$ rtg sv [OPTION]... -o DIR -t SDF -I FILE -R FILE
```

Example:

```
$ rtg sv -o sv_out -t genome -I sam-list.txt -R rgstats-list.txt
```

Parameters:

File Input/Output		
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
-o	--output=DIR	Directory for output.
	--readgroup-labels=FILE	File containing read group relabel mappings (1 per line with the format: [input_readgroup_id][tab][output_readgroup_id]).
-r	--readgroup-stats=FILE	Text file containing read group stats. May be specified 0 or more times.
-R	--readgroup-stats-list-file=FILE	File containing list of read group stats files (1 per line)
	--region=REGION	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
	--simple-signals	If set, also output simple signals.
-t	--template=SDF	SDF containing the reference genome.
	FILE+	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
-f	--fine-step=INT	Set the step size in interesting regions. (Default is 10).
-m	--max-as-mated=INT	Set to ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
-u	--max-as-unmated=INT	Set to ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
-s	--step=INT	Set the step size. (Default is 100).

Utility		
-h	--help	Prints help on command-line flag usage.
-Z	--no-gzip	Set this flag to create the output files without compression. By default the output files are compressed with tabix compatible blocked gzip.
-T	--threads=INT	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

This command takes as input a set of mappings and a reference genome. It applies Bayesian models to signals comprised of levels of mated, unmated and discordant mappings to predict the likelihood of various structural variant categories. The output of the `sv` command is in the form of two files: `sv_interesting.bed.gz` is a BED format file that identifies regions that potentially indicate a structural variant of some kind; `sv_bayesian.tsv.gz` is a tab separated format that contains the prediction strengths of each event model.

Table : Bayesian SV indicators

Indicator	Description
normal	No structural variant.
duplicate-left	The left border of a duplication.
duplicate	Position within a duplicated region.
duplicate-right	The right border of a duplication.
delete-left	The left border of a deletion.
delete	Position within a deletion.
delete-right	The right border of a deletion.
breakpoint	A breakpoint such as at the site where a duplicated section is inserted.
novel-insertion	A site receiving a novel insertion.

There are also heterozygous versions of each of these models.

The final column gives the index of the dominant hypothesis to allow easier extraction of sequences (for example a sequence of delete-left, delete, delete-right is a strong indicator of a deletion and can be used to identify the potential bounds of the deletion).

At this stage, analysis and filtering of the `sv` command output files is up to the end user.

The Bayesian `sv` command uses CPU proportional to the number of read groups, so it may be advantageous to merge related read groups (those that have the same read length and fragment size characteristics). Supplying a relabel file which maps every input read group to the same logical read group name would treat all alignments as if there were only one read group.

For additional information about the `sv` command output files see *SV command output file descriptions*.

See also:

svprep, *discord*

2.7.11 cnv

Synopsis:

The `cnv` command identifies copy number variation statistics and reports in a BED format file. Alignments for a test genome (typically a tumor sample) are compared to alignments for a base genome (typically a normal or matched control), and the ratios calculated.

Syntax:

Multi-file input specified from command line:

```
$ rtg cnv [OPTION]... -o DIR -i FILE -j FILE
```

Multi-file input specified in a text file:

```
$ rtg cnv [OPTION]... -o DIR -I FILE -J FILE
```

Example:

```
$ rtg cnv -b 1000 -o h1_cnv -i h1_base -j h1_test
```

Parameters:

File Input/Output		
-i	--base-file=FILE	SAM/BAM format files containing mapped reads for baseline. May be specified 0 or more times.
-I	--base-list-file=FILE	File containing list of SAM/BAM format files (1 per line) containing mapped reads for baseline.
-o	--output=DIR	Directory for output.
	--region=REGION	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
-t	--template=SDF	SDF containing the reference genome.
-j	--test-file=FILE	SAM/BAM format files containing mapped reads for test. May be specified 0 or more times.
-J	--test-list-file=FILE	File containing list of SAM/BAM format files (1 per line) containing mapped reads for test.

Sensitivity Tuning		
-b	--bucket-size=INT	Set size of the buckets in the genome. Use the bucket size to determine CNV coverage, bucket size defines the number of nucleotides to average the coverage for in a region. (Default is 100)
	--exclude-mated	Set to exclude all mated SAM records.
	--exclude-unmated	Set to exclude all unmated SAM records.
-m	--max-as-mated=INT	Set to ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
-u	--max-as-unmated=INT	Set to ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
-c	--max-hits=INT	Set to ignore SAM records with an alignment count that exceeds this value. This flag is usually set to 1 because an alignment count of 1 represents uniquely mapped reads.
	--min-mapq=INT	Set to ignore SAM records with MAPQ less than this value.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

The `cnv` command identifies aberrational CNV region(s) that support investigation of structural variation for WGS cancer sequencing data where a matched normal sample is available. It measures and reports the ratio of coverage depth in a test sample compared to a baseline sample. Use the `--bucket-size=INT` parameter to specify the range in which CNV ratios are calculated (for data smoothing). Filter settings allow different analytical comparisons with the same alignments.

See also:

snp, *coverage*

2.8 Metagenomics Commands

2.8.1 species

Synopsis:

Calculates a taxon distribution from a metagenomic sample.

Syntax:

Multi-file input specified from command line:

```
$ rtg species [OPTION]... -t SDF -o DIR FILE+
```

Multi-file input specified in a text file:

```
$ rtg species [OPTION]... -t SDF -o DIR -I FILE
```

Example:

```
$ rtg species -t genomes -o sp_out alignments.bam
```

Parameters:

File Input/Output		
-t	--genomes=SDF	SDF containing the genomes.
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
-o	--output=DIR	Directory for output.
-r	--relabel-species-file=FILE	File containing list of species name to reference name mappings (1 mapping per line format: [reference short name][tab][species])
	FILE+	SAM/BAM format files containing mapped reads. May be specified 0 or more times.

Sensitivity Tuning		
	--exclude-mated	Exclude all mated SAM records.
	--exclude-unmated	Exclude all unmated SAM records.
-m	--max-as-mated=INT	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
-u	--max-as-unmated=INT	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.

Reporting		
-c	--min-confidence=FLOAT	Species below this confidence value will not be reported (Default is 10.0)
	--print-all	Print non present species in the output file.

Utility		
-h	--help	Print help on command-line flag usage.
-T	--threads=INT	Number of threads (Default is the number of available cores)

Usage:

This command takes as input a set of SAM/BAM alignment data from a sample of DNA and a set of known genomes. It outputs an estimate of the fraction of the sample taken up by each of the genomes. For best results the reference SDF containing the genomes should be in the RTG taxonomic reference file format. Existing metagenomics reference SDFs in this format are available from our website (http://www.realtimegenomics.com). For more information about this format see *RTG taxonomic reference file format*.

When not using RTG taxonomic reference SDFs, if more than one sequence in the reference corresponds to the same species, use the `--relabel-species-file` flag to specify a file containing the mappings of short reference names to species names.

The `species` command assumes that the mappings of the sample against the reference species are well-modeled by a Poisson distribution. A multi-dimensional direct non-linear optimization procedure is used to minimize the error according to the Poisson model, leading to a posterior probability assignment for each of the reference sequences. The computation can account for stretches of reference sequence not appearing in the sample and for unmapped reads in the sample. So to get the best results, any unmapped reads should be included as part of the input.

The posterior probabilities are used to directly compute taxon representation in two ways. The first representation is the fractional abundance of particular taxon in the sample. The second representation is normalized to DNA size and reports the percentage of the particular DNA sequence in the sample.

Most of the columns in the `species.tsv` file are about estimating the abundance of particular species and clades. The output also contains a confidence score that addresses the subtly different question, "How likely is it that this species or clade is actually present in the sample?". The details of the calculation are somewhat complicated, but for a single species (more precisely, a leaf node in the taxonomy) the confidence is calculated as a log likelihood ratio between two posteriors. Internally, the species tool computes a posterior, , connected to the abundance estimate for a species, corresponding to the null hypothesis "species present at level P ". For an alternative hypothesis, corresponding to "species not present", another posterior, P' , is computed by forcing the estimated abundance for that species to 0. Confidence is then the log ratio of the two values, $C = ln(\frac{P'}{P})$. The number reported in the *confidence* column is $\sqrt{C}$. Taking the square root makes the units of confidence standard deviations and reduces the spread of values. By adjusting the `--min-confidence` parameter you can allow only results with a higher confidence to be output.

In addition to the raw output file, an interactive graphical view in HTML5 is also generated. Opening this shows the taxonomy and data on an interactive pie chart, with wedge sizes defined by either the abundance or DNA fraction (user selectable in the report).

See also:

similarity

2.8.2 similarity

Synopsis:

Produces a similarity matrix and nearest neighbor tree from the input sequences or reads.

Syntax:

Single-file genome per sequence input:

```
$ rtg similarity [OPTION]... -o DIR -i SDF
```

Multi-file genome per label input specified in a text file:

```
$ rtg similarity [OPTION]... -o DIR -I FILE
```

Example:

```
$ rtg similarity -o simil_out -i species_genomes
```

Parameters:

File Input/Output		
`-I`	`--input-list-file=FILE`	Specifies a file containing a labeled list of SDFs (one label and SDF per line format: [label][space][SDF])
`-i`	`--input=SDF`	Specifies the SDF containing a subject data set.
`-o`	`--output=DIR`	Specifies the directory where results are reported.

Sensitivity Tuning		
`-s`	`--step=INT`	Set the step size. (Default is 1).
	`--unique-words`	Set to count only unique words.
`-w`	`--word=INT`	Set the word size. (Default is 25).

Utility		
`-h`	`--help`	Print help on command-line flag usage.
	`--max-reads=INT`	Set the maximum number of reads to use from each input SDF. Use to reduce memory requirements in multi-file mode.

Usage:

Use in single-file mode to produce a similarity matrix and nearest neighbor tree where each sequence in the SDF is treated as a single genome for the comparisons. However, if the input SDF contains a taxonomy, then individual

sequences will be appropriately grouped in terms of the taxonomy and the resulting nearest neighbor tree will be in terms of the organisms of the taxonomy.

Use in multi-file mode to produce a similarity matrix and nearest neighbor tree for labeled sets of sequences where each label is treated as a single genome for the comparisons. The input file for this mode is of the form `[label][space][file]`, 1 per line where labels can be repeated to treat multiple SDFs as part of the same genome. Example:

```
SARS_coronavirus sars_sample1.sdf
SARS_coronavirus sars_sample2.sdf
Bacteriophage_KVP40 kvp40_sample1.sdf
Bacteriophage_KVP40 kvp40_sample2.sdf
```

The similarity tool outputs phylogenetic tree files, a similarity matrix file and a principal component analysis file. For more detail about the output files see *Species results file description*.

See also:

species

2.9 Pipeline Commands

RTG includes some pipeline commands that perform simple end-to-end tasks using a set of other RTG commands.

2.9.1 composition-meta-pipeline

Synopsis:

Runs a metagenomic composition pipeline. The pipeline consists of read filtering, read alignment, then species composition.

Syntax:

SDF or single-end FASTQ input:

```
$ rtg composition-meta-pipeline [OPTION]... --output DIR --input SDF|FILE
```

Paired-end FASTQ input:

```
$ rtg composition-meta-pipeline [OPTION]... --output DIR --input-left FILE \
  --input-right FILE
```

Example:

```
$ rtg composition-meta-pipeline --output comp_out --input bact_reads \
  --filter hg19 --species bact_db
```

Parameters:

File Input/Output		
	`--filter=SDF`	Specifies the SDF containing the filter reference sequences.
	`--input=SDF|FILE`	Specifies the path to the reads to be processed.
	`--input-left=FILE`	The left input file for FASTQ paired end reads.
	`--input-right=FILE`	The right input file for FASTQ paired end reads.
	`--output=DIR`	Specifies the directory where results are reported.
	`--platform`	Specifies the platform of the input data. (Must be one of [illumina, iontorrent]) (Default is illumina).
	`--species=SDF`	Specifies the SDF containing species reference sequences.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

The `composition-meta-pipeline` command runs a sequence of RTG commands to generate a species composition analysis from a set of input reads. Each command run outputs to a subdirectory within the output directory set with the `--output` flag.

The reads input data for this command must either be in SDF format, or be FASTQ files that use Sanger quality value encoding. If your data is not in this format, (e.g. FASTA or using Solexa quality value encoding), you should first create an SDF containing the reads using the `format` command, with appropriate command-line flags.

The reads are filtered to remove contaminant reads using the `mapf` command using the reference from the `--filter` flag. The `--sam-rg` flag of the `mapf` command is set with the platform specified by the `--platform` flag. If the input is given as FASTQ instead of in SDF format, the `--quality-format` is set to `sanger`. All other flags are left as the defaults defined in the `mapf` command description. The output subdirectory for the filter results is called `mapf`.

The unmapped reads from the read filtering step are aligned with the map command using the reference from the `--species` flag. The `--sam-rg` flag of the map command is set with the platform specified by the `--platform` flag. The `--max-mismatches` flag is set to 10% if the `--platform` flag is set to `illumina`, or 15% if set to `iontorrent`. The `--max-top-results` flag is set to 100. All other flags are left as the defaults defined in the `map` command description. The output subdirectory for the alignment results is called `map`.

The aligned reads are processed with the species command using the reference from the `--species` flag. Flag defaults defined in the species command description are used. The output subdirectory for the species composition results is called `species`.

A summary report about the results of all the steps involved is output to a subdirectory called `report`.

This pipeline command will use a default location for the reference SDF files when not specified explicitly on the command line. The default locations for each is within a subdirectory of the installation directory called `references`, with each SDF in the directory being the same name as the flag for it. For example the `--filter` flag will default to `/path/to/installation/references/filter`. To change the directory where it looks for these default references set the `RTG_REFERENCES_DIR` configuration property to the directory containing your default references (see *Advanced installation configuration*). Reference SDFs for use with the pipeline are available for download from our website (http://www.realtimegenomics.com).

See also:

mapf, *map*, *species*, *composition-functional-meta-pipeline*

2.9.2 functional-meta-pipeline

Synopsis:

Runs a metagenomic functional pipeline. The pipeline consists of read filtering, then protein searching.

Syntax:

SDF or single-end FASTQ input:

```
$ rtg functional-meta-pipeline [OPTION]... --output DIR --input SDF|FILE
```

Paired-end FASTQ input:

```
$ rtg functional-meta-pipeline [OPTION]... --output DIR --input-left FILE \
  --input-right FILE
```

Example:

```
$ rtg functional-meta-pipeline --output comp_out --input bact_reads --filter hg19 \
  --protein protein_db
```

Parameters:

File Input/Output		
	`--filter=SDF`	Specifies the SDF containing the filter reference sequences.
	`--input=SDF|FILE`	Specifies the path to the reads to be processed.
	`--input-left=FILE`	The left input file for FASTQ paired end reads.
	`--input-right=FILE`	The right input file for FASTQ paired end reads.
	`--output=DIR`	Specifies the directory where results are reported.
	`--platform`	Specifies the platform of the input data. Allowed values are [illumina, iontorrent] (Default is illumina)
	`--protein=SDF`	Specifies the SDF containing protein reference sequences.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

The `functional-meta-pipeline` command runs a sequence of RTG commands to generate a protein analysis from a set of input reads. Each command run outputs to a subdirectory within the output directory set with the `--output` flag.

The reads input data for this command must either be in SDF format, or be FASTQ files that use Sanger quality value encoding. If your data is not in this format, (e.g. FASTA or using Solexa quality value encoding), you should first create an SDF containing the reads using the `format` command, with appropriate command-line flags.

The reads are filtered to remove contaminant reads using the `mapf` command using the reference from the `--filter` flag. The `--sam-rg` flag of the `mapf` command is set with the platform specified by the `--platform` flag. If the input is given as FASTQ instead of in SDF format, the `--quality-format` is set to `sanger`. All other flags are left as the defaults defined in the `mapf` command description. The output subdirectory for the filter results is called `mapf`.

The unmapped reads from the read filtering step are processed with the `mapx` command using the reference from the `--protein` flag. The `--max-alignment-score` flag is set to 10% if the `--platform` flag is set to `illumina`, or 15% if set to `iontorrent`. The `--max-top-results` flag is set to 10. All other flags are left as the defaults defined in the `mapx` command description. If the input reads are single end the output will be to the `mapx1` subdirectory. If the input reads are paired end, the reads from each end are processed separately. The output for the left end will be the `mapx1` subdirectory and for the right end will be the `mapx2` subdirectory.

A summary report about the results of all the steps involved is output to a subdirectory called `report`.

This pipeline command will use a default location for the reference SDF files when not specified explicitly on the command line. The default locations for each is within a subdirectory of the installation directory called `references`, with each SDF in the directory being the same name as the flag for it. For example the `--filter` flag will default to `/path/to/installation/references/filter`. To change the directory where it looks for these default references set the `RTG_REFERENCES_DIR` configuration property to the directory containing your default references (see *Advanced installation configuration*). Reference SDFs for use with the pipeline are available for download from our website (http://www.realtimegenomics.com).

See also:

mapf, *mapx*, *composition-functional-meta-pipeline*

2.9.3 composition-functional-meta-pipeline

Synopsis:

Runs the metagenomic composition and functional pipelines. The pipelines consist of read filtering, read alignment then species composition, and protein searching.

Syntax:

SDF or single-end FASTQ input:

```
$ rtg composition-functional-meta-pipeline [OPTION]... --output DIR \
  --input SDF|FILE
```

Paired-end FASTQ input:

```
$ rtg composition-functional-meta-pipeline [OPTION]... --output DIR \
  --input-left FILE --input-right FILE
```

Example:

```
$ rtg composition-functional-meta-pipeline --output comp_out --input bact_reads \
  --filter hg19 --species bact_db --protein protein_db
```

Parameters:

File Input/Output		
	`--filter=SDF`	Specifies the SDF containing the filter reference sequences.
	`--input=SDF|FILE`	Specifies the path to the reads to be processed.
	`--input-left=FILE`	The left input file for FASTQ paired end reads.
	`--input-right=FILE`	The right input file for FASTQ paired end reads.
	`--output=DIR`	Specifies the directory where results are reported.
	`--platform`	Specifies the platform of the input data. Allowed values are [illumina, iontorrent] (Default is illumina).
	`--species=SDF`	Specifies the SDF containing species reference sequences.
	`--protein=SDF`	Specifies the SDF containing protein reference sequences.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

The `composition-functional-meta-pipeline` command runs a sequence of RTG commands to generate a species composition analysis and a protein analysis from a set of input reads. Each command run outputs to a subdirectory within the output directory set with the `--output` flag.

The reads input data for this command must either be in SDF format, or be FASTQ files that use Sanger quality value encoding. If your data is not in this format, (e.g. FASTA or using Solexa quality value encoding), you should first create an SDF containing the reads using the `format` command, with appropriate command-line flags.

The reads are filtered to remove contaminant reads using the `mapf` command using the reference from the `--filter` flag. The `--sam-rg` flag of the `mapf` command is set with the platform specified by the `--platform` flag. If the input is given as FASTQ instead of in SDF format, the `--quality-format` is set to `sanger`. All other flags are left as the defaults defined in the `mapf` command description. The output subdirectory for the filter results is called `mapf`.

The unmapped reads from the read filtering step are aligned with the `map` command using the reference from the `--species` flag. The `--sam-rg` flag of the `map` command is set with the platform specified by the `--platform` flag. The `--max-mismatches` flag is set to 10% if the `--platform` flag is set to `illumina`, or 15% if set to `iontorrent`. The `--max-top-results` flag is set to 100. All other flags are left as the defaults defined in the `map` command description. The output subdirectory for the alignment results is called `map`.

The aligned reads are processed with the `species` command using the reference from the `--species` flag. Flag defaults defined in the `species` command description are used. The output subdirectory for the species composition results is called `species`.

The unmapped reads from the read filtering step are processed with the `mapx` command using the reference from the `--protein` flag. The `--max-alignment-score` flag is set to 10% if the `--platform` flag is set to `illumina`, or 15% if set to `iontorrent`. The `--max-top-results` flag is set to 10. All other flags are left as the defaults defined in the `mapx` command description. If the input reads are single end the output will be to the `mapx1` subdirectory. If the input reads are paired end, the reads from each end are processed separately. The output for the left end will be the `mapx1` subdirectory and for the right end will be the `mapx2` subdirectory.

A summary report about the results of all the steps involved is output to a subdirectory called `report`.

This pipeline command will use a default location for the reference SDF files when not specified explicitly on the command line. The default locations for each is within a subdirectory of the installation directory called `references`, with each SDF in the directory being the same name as the flag for it. For example the `--filter`

flag will default to `/path/to/installation/references/filter`. To change the directory where it looks for these default references set the `RTG_REFERENCES_DIR` configuration property to the directory containing your default references (see *Advanced installation configuration*). Reference SDFs for use with the pipeline are available for download from our website (http://www.realtimegenomics.com).

See also:

mapf, *mapx*, *map*, *species*

2.10 Simulation Commands

RTG includes some simulation commands that may be useful for experimenting with effects of various RTG command parameters or when getting familiar with RTG work flows. A simple simulation series might involve the following commands:

```
$ rtg genomesim --output sim-ref-sdf --min-length 500000 --max-length 5000000 \
  --num-contigs 5
$ rtg popsim --reference sim-ref-sdf --output population.vcf.gz
$ rtg samplesim --input population.vcf.gz --output sample1.vcf.gz \
  --output-sdf sample1-sdf --reference sim-ref-sdf --sample sample1
$ rtg readsim --input sample1-sdf --output reads-sdf --machine illumina_pe \
  -L 75 -R 75 --coverage 10
$ rtg map --template sim-ref-sdf --input reads-sdf --output sim-mapping \
  --sam-rg "@RG\tID:sim-rg\tSM:sample1\tPL:ILLUMINA"
$ rtg snp --template sim-ref-sdf --output sim-name-snp sim-mapping/alignments.bam
```

2.10.1 genomesim

Synopsis:

Use the `genomesim` command to simulate a reference genome, or to create a baseline reference genome for a research project when an actual genome reference sequence is unavailable.

Syntax:

Specify number of sequences, plus minimum and maximum lengths:

```
$ rtg genomesim [OPTION]... -o SDF --max-length INT --min-length INT -n INT
```

Specify explicit sequence lengths (one more sequences):

```
$ rtg genomesim [OPTION]... -o SDF -l INT
```

Example:

```
$ rtg genomesim -o genomeTest -l 500000
```

Parameters:

File Input/Output		
`-o`	`--output=SDF`	The name of the output SDF.

Utility		
	`--comment=STRING`	Specify a comment to include in the generated SDF.
	`--freq=STRING`	Set the relative frequencies of A,C,G,T in the generated sequence. (Default is 1,1,1,1).
`-h`	`--help`	Prints help on command-line flag usage.
`-l`	`--length=INT`	Specify the length of generated sequence. May be specified 0 or more times, or as a comma separated list.
	`--max-length=INT`	Specify the maximum sequence length.
	`--min-length=INT`	Specify the minimum sequence length.
`-n`	`--num-contigs=INT`	Specify the number of sequences to generate.
	`--prefix=STRING`	Specify a sequence name prefix to be used for the generated sequences. The default is to name the output sequences 'simulatedSequenceN', where N is increasing for each sequence.
`-s`	`--seed=INT`	Specify seed for the random number generator.

Usage:

The `genomesim` command allows one to create a simulated genome with one or more contiguous sequences - exact lengths of each contig or number of contigs with minimum and maximum lengths provided. The contents of an SDF directory created by `genomesim` can be exported to a FASTA file using the `sdf2fasta` command.

This command is primarily useful for providing a simple randomly generated base genome for use with subsequent simulation commands.

Each generated contig is named by appending an increasing numeric index to the specified prefix. For example `--prefix=chr --num-contigs=10` would yield contigs named `chr1` through `chr10`.

See also:

cgsim, *readsim*, *popsim*, *samplesim*

2.10.2 cgsim

Synopsis:

Simulate Complete Genomics Inc sequencing reads. Supports the original 35 bp read structure (5-10-10-10), and the newer 29 bp read structure (10-9-10).

Syntax:

Generation by genomic coverage multiplier:

```
$ rtg cgsim [OPTION]... -V INT -t SDF -o SDF -c FLOAT
```

Generation by explicit number of reads:

```
$ rtg cgsim [OPTION]... -V INT -t SDF -o SDF -n INT
```

Example:

```
$ rtg cgsim -V 1 -t HUMAN_reference -o CG_3x_readst -c 3
```

Parameters:

File Input/Output		
`-t`	`--input=SDF`	SDF containing input genome.
`-o`	`--output=SDF`	Name for reads output SDF.

Fragment Generation		
	`--abundance`	If set, the user-supplied distribution represents desired abundance.
`-N`	`--allow-unknowns`	Allow reads to be drawn from template fragments containing unknown nucleotides.
`-c`	`--coverage=FLOAT`	Coverage, must be positive.
`-D`	`--distribution=FILE`	File containing probability distribution for sequence selection.
	`--dna-fraction`	If set, the user-supplied distribution represents desired DNA fraction.
`-M`	`--max-fragment-size=INT`	Maximum fragment size (Default is 500)
`-m`	`--min-fragment-size=INT`	Minimum fragment size (Default is 350)
	`--n-rate=FLOAT`	Rate that the machine will generate new unknowns in the read (Default is 0.0)
`-n`	`--num-reads=INT`	Number of reads to be generated.
	`--taxonomy-distribution=FILE`	File containing probability distribution for sequence selection expressed by taxonomy id.

Complete Genomics		
`-V`	`--cg-read-version=INT`	Select Complete Genomics read structure version, 1 (35 bp) or 2 (29 bp)

Utility		
	`--comment=STRING`	Comment to include in the generated SDF.
`-h`	`--help`	Print help on command-line flag usage.
	`--no-names`	Do not create read names in the output SDF.
	`--no-qualities`	Do not create read qualities in the output SDF.
`-q`	`--qual-range=STRING`	Set the range of base quality values permitted e.g.: 3-40 (Default is fixed qualities corresponding to overall machine base error rate)
	`--sam-rg=STRING|FILE`	File containing a single valid read group SAM header line or a string in the form `@RG\tID:READGROUP1\tSM:BACT_SAMPLE\tPL:ILLUMINA`
`-s`	`--seed=INT`	Seed for random number generator.

Usage:

Use the `cgsim` command to set either `--coverage` or `--num-reads` in simulated Complete Genomics reads. For more information about Complete Genomics reads, refer to http://www.completegenomics.com

RTG simulation tools allows for deterministic experiment repetition. The `--seed` parameter, for example, allows for regeneration of exact same reads by setting the random number generator to be repeatable (without supplying this flag a different set of reads will be generated each time).

The `--distribution` parameter allows you to specify the probability that a read will come from a particular named sequence for use with metagenomic databases. Probabilities are numbers between zero and one and must sum to one in the file.

See also:

genomesim, *readsim*, *popsim*, *samplesim*

2.10.3 readsim

Synopsis:

Use the `readsim` command to generate single or paired end reads of fixed or variable length from a reference genome, introducing machine errors.

Syntax:

Generation by genomic coverage multiplier:

```
$ rtg readsim [OPTION]... -t SDF --machine STRING -o SDF -c FLOAT
```

Generation by explicit number of reads:

```
$ rtg readsim [OPTION]... -t SDF --machine STRING -o SDF -n INT
```

Example:

```
$ rtg readsim -t genome_ref -o sim_reads -r 75 --machine illumina_se  -c 30
```

Parameters:

File Input/Output		
-t	--input=SDF	SDF containing input genome.
	--machine=STRING	Select the sequencing technology to model. Allowed values are [illumina_se, illumina_pe, complete_genomics, complete_genomics_2, 454_pe, 454_se, iontorrent]
-o	--output=SDF	Name for reads output SDF.

Fragment Generation		
	--abundance	If set, the user-supplied distribution represents desired abundance.
-N	--allow-unknowns	Allow reads to be drawn from template fragments containing unknown nucleotides.
-c	--coverage=FLOAT	Coverage, must be positive.
-D	--distribution=FILE	File containing probability distribution for sequence selection.
	--dna-fraction	If set, the user-supplied distribution represents desired DNA fraction.
-M	--max-fragment-size=INT	Maximum fragment size (Default is 250)
-m	--min-fragment-size=INT	Minimum fragment size (Default is 200)
	--n-rate=FLOAT	Rate that the machine will generate new unknowns in the read (Default is 0.0)
-n	--num-reads=INT	Number of reads to be generated.
	--taxonomy-distribution=FILE	File containing probability distribution for sequence selection expressed by taxonomy id.

Illumina PE		
-L	--left-read-length=INT	Target read length on the left side.
-R	--right-read-length=INT	Target read length on the right side.

Illumina SE		
-r	--read-length=INT	Target read length, must be positive.

454 SE/PE		
	--454-max-total-size=INT	Maximum 454 read length (in paired end case the sum of the left and the right read lengths)
	--454-min-total-size=INT	Minimum 454 read length (in paired end case the sum of the left and the right read lengths)

IonTorrent SE		
	--ion-max-total-size=INT	Maximum IonTorrent read length.
	--ion-min-total-size=INT	Minimum IonTorrent read length.

Utility		
	`--comment=STRING`	Comment to include in the generated SDF.
`-h`	`--help`	Print help on command-line flag usage.
	`--no-names`	Do not create read names in the output SDF.
	`--no-qualities`	Do not create read qualities in the output SDF.
`-q`	`--qual-range=STRING`	Set the range of base quality values permitted e.g.: 3-40 (Default is fixed qualities corresponding to overall machine base error rate)
	`--sam-rg=STRING|FILE`	File containing a single valid read group SAM header line or a string in the form `@RG\tID:READGROUP1\tSM:BACT_SAMPLE\tPL:ILLUMINA`
`-s`	`--seed=INT`	Seed for random number generator.

Usage:

Create simulated reads from a reference genome by either specifying coverage depth or a total number of reads.

A typical use case involves creating a mutated genome by introducing SNPs or CNVs with `popsim` and `samplesim` generating reads from the mutated genome with `readsim`, and mapping them back to the original reference to verify the parameters used for mapping or variant detection.

RTG simulation tools allows for deterministic experiment repetition. The `--seed` parameter, for example, allows for regeneration of exact same reads by setting the random number generator to be repeatable (without supplying this flag a different set of reads will be generated each time).

The `--distribution` parameter allows you to specify the sequence composition of the resulting read set, primarily for use with metagenomic databases. The distribution file is a text file containing lines of the form:

```
<probability><space><sequence name>
```

Probabilities must be between zero and one and must sum to one in the file. For reference databases containing taxonomy information, where each species may be comprised of more than one sequence, it is instead possible to use the `--taxonomy-distribution` option to specify the probabilities at a per-species level. The format of each line in this case is:

```
<probability><space><taxon id>
```

When using `--distribution` or `--taxonomy-distribution`, the interpretation must be specified one of `--abundance` or `--dna-fraction`. When using `--abundance` each specified probability reflects the chance of selecting the specified sequence (or taxon id) from the set of sequences, and thus for a given abundance a large sequence will be represented by more reads in the resulting set than a short sequence. In contrast, with `--dna-fraction` each specified probability reflects the chance of a read being derived from the designated sequence, and thus for a given fraction, a large sequence will have a lower depth of coverage than a short sequence.

See also:

cgsim, *genomesim*, *popsim*, *samplesim*

2.10.4 readsimeval

Synopsis:

Use the `readsimeval` command to examine the mapping accuracy of reads previously generated by the `readsim` command.

Syntax:

```
$ rtg readsimeval [OPTION]... -o DIR -r SDF FILE+
```

Example:

```
$ rtg readsimeval -t genome_ref -o map_rse -r reads_sd map/alignments.bam
```

Parameters:

File Input/Output		
-M	--mutations-vcf=FILE	VCF file containing genomic mutations to be compensated for.
-o	--output=DIR	Directory for output.
-r	--reads=SDF	SDF containing reads.
-S	--sample=STRING	Name of the sample to use from the mutation VCF file, will default to using the first sample in the file.
	FILE+	SAM/BAM format files. Must be specified 1 or more times.

Sensitivity Tuning		
	--exclude-duplicates	Exclude all SAM records flagged as a PCR or optical duplicate.
	--exclude-mated	Exclude all mated SAM records.
	--exclude-unmated	Exclude all unmated SAM records.
	--max-as-mated=INT	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
	--max-as-unmated=INT	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
	--min-mapq=INT	If set, ignore SAM records with MAPQ less than this value.
-v	--variance=INT	Variation allowed in start position (Default is 0).

Reporting		
	--mapq-histogram	Output histogram of MAPQ scores.
	--mapq-roc	Output ROC table with respect to MAPQ scores.
	--score-histogram	Output histogram of read alignment / generated scores.
	--verbose	Provide more detailed breakdown of stats.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

This command can be used to evaluate the mapping accuracy on reads that have been generated by the `readsim` command. The ROC output files may be plotted with the `rocplot` command.

See also:

cgsim, *readsim*, *rocplot*

2.10.5 popsim

Synopsis:

Use the `popsim` command to generate a VCF containing simulated population variants. Each variant allele generated has an associated frequency `INFO` field describing how frequent in the population that allele is.

Syntax:

```
$ rtg popsim [OPTION]... -o FILE -t SDF
```

Example:

```
$ rtg popsim -o pop.vcf -t HUMAN_reference
```

Parameters:

File Input/Output		
-o	--output=FILE	Output VCF file name.
-t	--reference=SDF	SDF containing the reference genome.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--seed=INT	Seed for the random number generator.

Usage:

The popsim command is used to create a VCF containing variants with frequency in population information that can be subsequently used to simulate individual samples using the samplesim command. The frequency in population is contained in a VCF INFO field called AF. The types of variants and the allele-frequency distribution has been drawn from observed variants and allele frequency distribution in human studies.

See also:

readsim, *genomesim*, *samplesim*, *childsim*, *samplereplay*

2.10.6 samplesim

Synopsis:

Use the samplesim command to generate a VCF containing a genotype simulated from population variants according to allele frequency.

Syntax:

```
$ rtg samplesim [OPTION]... -i FILE -o FILE -t SDF -s STRING
```

Example:

From a population frequency VCF:

```
$ rtg samplesim -i pop.vcf -o 1samples.vcf -t HUMAN_reference -s person1 --sex male
```

From an existing simulated VCF:

```
$ rtg samplesim -i 1samples.vcf -o 2samples.vcf -t HUMAN_reference -s person2 \
  --sex female
```

Parameters:

File Input/Output		
-i	--input=FILE	Input VCF containing population variants.
-o	--output=FILE	Output VCF file name.
	--output-sdf=SDF	If set, output an SDF containing the sample genome.
-t	--reference=SDF	SDF containing the reference genome.
-s	--sample=STRING	Name for sample.

Utility		
	--allow-missing-af	If set, treat variants without allele frequency annotation as uniformly likely.
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--ploidy=STRING	Ploidy to use. Allowed values are [auto, diploid, haploid] (Default is auto)
	--seed=INT	Seed for the random number generator.
	--sex=SEX	Sex of individual. Allowed values are [male, female, either] (Default is either)

Usage:

The samplesim command is used to simulate an individuals genotype information from a population variant frequency VCF generated by the popsim command or by previous samplesim or childsim commands. The new output VCF will contain all the existing variants and samples with a new column for the new sample. The

genotype at each record of the VCF will be chosen randomly according to the allele frequency specified in the `AF` field.

If input VCF records do not contain an `AF` annotation, by default any ALT allele in that record will not be selected and so the sample will be genotyped as 0/0. Alternatively for simple simulations the `--allow-missing-af` flag will treat each allele in such records as being equally likely (i.e.: effectively equivalent to `AF=0.5` for a biallelic variant, `AF=0.33,0.33` for a triallelic variant, etc).

The ploidy for each genotype is automatically determined according to the ploidy of that chromosome for the specified sex of the individual, as defined in the reference genome `reference.txt` file. For more information see *RTG reference file format*. If the reference SDF does not contain chromosome configuration information, a default ploidy can be specified using the `--ploidy` flag.

The `--output-sdf` flag can be used to optionally generate an SDF of the individuals genotype which can then be used by the `readsim` command to simulate a read set for the individual.

See also:

readsim, *genomesim*, *popsim*, *childsim*, *samplereplay*

2.10.7 denovosim

Synopsis:

Use the `denovosim` command to generate a VCF containing a derived genotype containing *de novo* variants.

Syntax:

```
$ rtg denovosim [OPTION]... -i FILE --original STRING -o FILE -t SDF -s STRING
```

Example:

```
$ rtg denovosim -i sample.vcf --original personA -o 2samples.vcf \
  -t HUMAN_reference -s personB
```

Parameters:

File Input/Output		
`-i`	`--input=FILE`	The input VCF containing parent variants.
	`--original=STRING`	The name of the existing sample to use as the original genotype.
`-o`	`--output=FILE`	The output VCF file name.
	`--output-sdf=FILE`	Set to output an SDF of the genome generated.
`-t`	`--reference=SDF`	The SDF containing the reference genome.
`-s`	`--sample=STRING`	The name for the new derived sample.

Utility		
`-h`	`--help`	Prints help on command-line flag usage.
`-Z`	`--no-gzip`	Set this flag to create the VCF output file without compression.
	`--num-mutations=INT`	Set the expected number of de novo mutations per genome (Default is 70).
	`--ploidy=STRING`	The ploidy to use when the reference genome does not contain a reference text file. Allowed values are [auto, diploid, haploid] (Default is auto)
	`--seed=INT`	Set the seed for the random number generator.
	`--show-mutations`	Set this flag to display information regarding de novo mutation points.

Usage:

The `denovosim` command is used to simulate a derived genotype containing *de novo* variants from a VCF containing an existing genotype.

The output VCF will contain all the existing variants and samples, along with additional *de novo* variants. If the original and derived sample names are different, the output will contain a new column for the mutated sample. If

the original and derived sample names are the same, the sample in the output VCF is updated rather than creating an entirely new sample column. When a sample receives a *de novo* mutation, the sample `DN` field is set to "Y".

If *de novo* variants were introduced without regard to neighboring variants, a situation could arise where it is not possible to unambiguously determine the haplotype of the simulated sample. To prevent this, `denovosim` will not output a *de novo* variant that overlaps existing variants. Since `denovosim` chooses candidate *de novo* locations before reading the input VCF, this occasionally mandates skipping a candidate *de novo* so the target number of mutations may not always be reached.

The `--output-sdf` flag can be used to optionally generate an SDF of the derived genome which can then be used by the `readsim` command to simulate a read set for the new genome.

See also:

readsim, *genomesim*, *popsim*, *samplesim*, *samplereplay*

2.10.8 childsim

Synopsis:

Use the `childsim` command to generate a VCF containing a genotype simulated as a child of two parents.

Syntax:

```
$ rtg childsim [OPTION]... --father STRING -i FILE --mother STRING -o FILE -t SDF \
  -s STRING
```

Example:

```
$ rtg childsim --father person1 --mother person2 -i 2samples.vcf -o 3samples.vcf \
  -t HUMAN_reference -s person3
```

Parameters:

File Input/Output		
	`--father=STRING`	Name of the existing sample to use as the father.
`-i`	`--input=FILE`	Input VCF containing parent variants.
	`--mother=STRING`	Name of the existing sample to use as the mother.
`-o`	`--output=FILE`	Output VCF file name.
	`--output-sdf=SDF`	If set, output an SDF containing the sample genome.
`-t`	`--reference=SDF`	SDF containing the reference genome.
`-s`	`--sample=STRING`	Name for new child sample.

Utility		
	`--extra-crossovers=FLOAT`	Probability of extra crossovers per chromosome (Default is 0.01)
`-h`	`--help`	Print help on command-line flag usage.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--ploidy=STRING`	Ploidy to use. Allowed values are [auto, diploid, haploid] (Default is auto)
	`--seed=INT`	Seed for the random number generator.
	`--sex=SEX`	Sex of individual. Allowed values are [male, female, either] (Default is either)
	`--show-crossovers`	If set, display information regarding haplotype selection and crossover points.

Usage:

The `childsim` command is used to simulate an individuals genotype information from a VCF containing the two parent genotypes generated by previous `samplesim` or `childsim` commands. The new output VCF will contain all the existing variants and samples with a new column for the new sample.

The ploidy for each genotype is generated according to the ploidy of that chromosome for the specified sex of the individual, as defined in the reference genome `reference.txt` file. For more information see *RTG reference*

file format. The generated genotypes are all consistent with Mendelian inheritance (*de novo* variants can be simulated with the `denovosim` command).

The `--output-sdf` flag can be used to optionally generate an SDF of the child's genotype which can then be used by the `readsim` command to simulate a read set for the child.

See also:

readsim, *genomesim*, *popsim*, *samplesim*, *samplereplay*

2.10.9 pedsamplesim

Synopsis:

Generates simulated genotypes for all members of a pedigree. `pedsamplesim` automatically simulates founder individuals, inheritance by children, and de novo mutations.

Syntax:

```
$ rtg pedsamplesim [OPTION]... -i FILE -o DIR -p FILE -t SDF
```

Example:

```
$ rtg pedsamplesim -t reference.sdf -p family.ped -i popvars.vcf \
  -o family_sim --remove-unused
```

Parameters:

File Input/Output		
`-i`	`--input=FILE`	Input VCF containing parent variants.
`-o`	`--output=DIR`	Directory for output.
	`--output-sdf`	If set, output an SDF for the genome of each simulated sample.
`-p`	`--pedigree=FILE`	Genome relationships PED file.
`-t`	`--reference=SDF`	SDF containing the reference genome.

Utility		
	`--extra-crossovers=FLOAT`	Probability of extra crossovers per chromosome (Default is 0.01)
`-h`	`--help`	Print help on command-line flag usage.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--num-mutations=INT`	Expected number of mutations per genome (Default is 70)
	`--ploidy=STRING`	Ploidy to use. Allowed values are [auto, diploid, haploid] (Default is auto)
	`--remove-unused`	If set, output only variants used by at least one sample.
	`--seed=INT`	Seed for the random number generator.

Usage:

The `pedsamplesim` uses the methods of `samplesim`, `denovosim`, and `childsim` to greatly ease the simulation of multiple samples. The input VCF should contain standard allele frequency INFO annotations that will be used to simulate genotypes for any sample identified as a founder. Any samples present in the pedigree that are already present in the input VCF will not be regenerated. To simulate genotypes for a subset of the members of the pedigree, use `pedfilter` to create a filtered pedigree file that includes only the subset required.

The supplied pedigree file is first examined to identify any individuals that cannot be simulated according to inheritance from other samples in the pedigree. Note that simulation according to inheritance requires both parents to be present in the pedigree. These samples in the pedigree are treated as founder individuals.

Founder individuals are simulated using `samplesim`, where the genotypes are chosen according to the allele frequency annotation in the input VCF.

All newly generated samples may have *de novo* mutations introduced according to the `--num-mutations` setting. As with the `denovosim` command, any *de novo* mutations introduced in a sample will be genotyped as homozygous reference in other pre-existing samples, and introduced variants will not overlap any pre-existing variant loci.

Samples that can be simulated according to Mendelian inheritance are then generated, using `childsim`. As expected, as well as inheriting *de novo* variants from parents, each child may obtain new *de novo* mutations of their own.

If the simulated samples will be used for subsequent simulated sequencing, such as via `readsim`, it is possible to automatically output an SDF containing the simulated genome for each sample by specifying the `--output-sdf` option, obviating the need to separately use `samplereplay`.

See also:

pedfilter, *popsim*, *samplesim*, *childsim*, *denovosim*, *samplereplay*, *readsim*

2.10.10 samplereplay

Synopsis:

Use the `samplereplay` command to generate the genome SDF corresponding to a sample genotype in a VCF file.

Syntax:

```
$ rtg samplereplay [OPTION]... -i FILE -o SDF -t SDF -s STRING
```

Example:

```
$ rtg samplereplay -i 3samples.vcf -o child.sdf -t HUMAN_reference -s person3
```

Parameters:

File Input/Output		
`-i`	`--input=FILE`	Input VCF containing the sample genotype.
`-o`	`--output=SDF`	Name for output SDF.
`-t`	`--reference=SDF`	SDF containing the reference genome.
`-s`	`--sample=STRING`	Name of the sample to select from the VCF.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

The `samplereplay` command can be used to generate an SDF of a genotype for a given sample from an existing VCF file. This can be used to generate a genome from the outputs of the `samplesim` and `childsim` commands. The output genome can then be used in simulating a read set for the sample using the `readsim` command.

Every chromosome for which the individual is diploid will have two sequences in the resulting SDF.

See also:

readsim, *genomesim*, *popsim*, *samplesim*, *childsim*

2.11 Utility Commands

2.11.1 bgzip

Synopsis:

Block compress a file or decompress a block compressed file. Block compressed outputs from the mapping and variant detection commands can be indexed with the `index` command. They can also be processed with standard gzip tools such as `gunzip` and `zcat`.

Syntax:

```
$ rtg bgzip [OPTION]... FILE+
```

Example:

```
$ rtg bgzip alignments.sam
```

Parameters:

File Input/Output		
-l	--compression-level=INT	The compression level to use, between 1 (least but fast) and 9 (highest but slow) (Default is 5)
-d	--decompress	Decompress.
-f	--force	Force overwrite of output file.
	--no-terminate	If set, do not add the block gzip termination block.
-c	--stdout	Write on standard output, keep original files unchanged. Implied when using standard input.
	FILE+	File to (de)compress, use '-' for standard input. Must be specified 1 or more times.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

Use the `bgzip` command to block compress files. Files such as VCF, BED, SAM, TSV must be block-compressed before they can be indexed for fast retrieval of records corresponding to specific genomic regions.

See also:

index

2.11.2 index

Synopsis:

Create tabix index files for block compressed TAB-delimited genome position data files or BAM index files for BAM files.

Syntax:

Multi-file input specified from command line:

```
$ rtg index [OPTION]... FILE+
```

Multi-file input specified in a text file:

```
$ rtg index [OPTION]... -I FILE
```

Example:

```
$ rtg index -f sam alignments.sam.gz
```

Parameters:

File Input/Output		
-f	--format=FORMAT	Format of input to index. Allowed values are [sam, bam, cram, sv, coveragetsv, bed, vcf, auto] (Default is auto)
-I	--input-list-file=FILE	File containing a list of block compressed files (1 per line) containing genome position data.
	FILE+	Block compressed files containing data to be indexed. May be specified 0 or more times.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

Use the `index` command to produce tabix indexes for block compressed genome position data files like SAM files, VCF files, BED files, and the TSV output from RTG commands such as `coverage`. The `index` command can also be used to produce BAM indexes for BAM files with no index.

See also:

map, *coverage*, *snp*, *extract*, *bgzip*

2.11.3 extract

Synopsis:

Extract specified parts of an indexed block compressed genome position data file.

Syntax:

Extract whole file:

```
$ rtg extract [OPTION]... FILE
```

Extract specific regions:

```
$ rtg extract [OPTION]... FILE STRING+
```

Example:

```
$ rtg extract alignments.bam 'chr1:2500000~1000'
```

Parameters:

File Input/Output		
	FILE	The indexed block compressed genome position data file to extract.

Filtering		
	REGION+	The range to display. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>. May be specified 0 or more times.

Reporting		
	--header	Set to also display the file header.
	--header-only	Set to only display the file header.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

Use the `extract` command to view specific parts of indexed block compressed genome position data files such as those in SAM/BAM/BED/VCF format.

See also:

map, *coverage*, *snp*, *index*, *bgzip*

2.11.4 aview

Synopsis:

View read mapping and variants corresponding to a region of the genome, with output as ASCII to the terminal, or HTML.

Syntax:

```
$ rtg aview [OPTION]... --region STRING -t SDF FILE+
```

Example:

```
$ rtg aview -t hg19 -b omni.vcf -c calls.vcf map/alignments.bam \
  --region Chr10:100000+3 -padding 30
```

Parameters:

File Input/Output		
-b	--baseline=FILE	VCF file containing baseline variants.
-B	--bed=FILE	BED file containing regions to overlay. May be specified 0 or more times.
-c	--calls=FILE	VCF file containing called variants. May be specified 0 or more times.
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line)
-r	--reads=SDF	Read SDF (only needed to indicate correctness of simulated read mappings). May be specified 0 or more times.
-t	--template=SDF	SDF containing the reference genome.
	FILE+	Alignment SAM/BAM files. May be specified 0 or more times.

Filtering		
-p	--padding=INT	Padding around region of interest (Default is to automatically determine padding to avoid read truncation)
	--region=REGION	The region of interest to display. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
	--sample=STRING	Specify name of sample to select. May be specified 0 or more times, or as a comma separated list.

Reporting		
	--html	Output as HTML.
	--no-base-colors	Do not use base-colors.
	--no-color	Do not use colors.
	--no-dots	Display nucleotide instead of dots.
	--print-cigars	Print alignment cigars.
	--print-mapq	Print alignment MAPQ values.
	--print-mate-position	Print mate position.
	--print-names	Print read names.
	--print-readgroup	Print read group id for each alignment.
	--print-reference-line=INT	Print reference line every N lines (Default is 0)
	--print-sample	Print sample id for each alignment.
	--print-soft-clipped-bases	Print soft clipped bases.
	--project-track=INT	If set, project highlighting for the specified track down through reads (Default projects the union of tracks)
	--sort-readgroup	Sort reads first on read group and then on start position.
	--sort-reads	Sort reads on start position.
	--sort-sample	Sort reads first on sample id and then on start position.
	--unflatten	Display unflattened CGI reads when present.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

Use the `aview` command to display a textual view of mappings and variants corresponding to a small region of the reference genome. This is useful when examining evidence for variant calls in a server environment where a graphical display application such as IGV is not available. The `aview` command is easy to script in order to output displays for multiple regions for later viewing (either as text or HTML).

See also:

map, *snp*

2.11.5 sdfstats

Synopsis:

Print statistics that describe a directory of SDF formatted data.

Syntax:

```
$ rtg sdfstats [OPTION]... SDF+
```

Example:

```
$ rtg sdfstats human_READS_SDF

Location            : C:\human_READS_SDF
Parameters          : format -f solexa -o human_READS_SDF
                                  c:\users\Elle\human\SRR005490.fastq.gz
SDF Version         : 6
Type                : DNA
Source              : SOLEXA
Paired arm          : UNKNOWN
Number of sequences: 4193903
Maximum length      : 48
Minimum length      : 48
N                   : 931268
A                   : 61100096
C                   : 41452181
G                   : 45262380
T                   : 52561419
Total residues      : 201307344
Quality scores available on this SDF
```

Parameters:

File Input/Output		
	SDF+	Specifies an SDF on which statistics are to be reported. May be specified 1 or more times.

Reporting		
	--lengths	Set to print out the name and length of each sequence. (Not recommended for read sets).
-p	--position	Set to include information about unknown bases (Ns) by read position.
-q	--quality	Set to display mean of quality.
	--sex=SEX	Set to display the reference sequence list for the given sex. Allowed values are [male, female, either]. May be specified 0 or more times, or as a comma separated list.
	--taxonomy	Set to display information about the taxonomy.
-n	--unknowns	Set to include information about unknown bases (Ns).

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

Use the `sdfstats` command to get information about the contents of SDFs.

See also:

format, *cg2sdf* , *sdf2fasta*, *sdf2fastq*, *sdfstats*, *sdfsplit*

2.11.6 sdfsplit

Synopsis:

Split SDF data into multiple equal segments, for parallel processing on a computer cluster when running commands that do not directly support processing a subset of a data set.

Syntax:

Command line SDF list:

```
$ rtg sdfsplit [OPTION]... -n INT -o DIR SDF+
```

File-based SDF list:

```
$ rtg sdfsplit [OPTION]... -n INT -o DIR -I FILE
```

Example:

```
$ rtg sdfsplit -n 260000 reads -o split_reads
```

Parameters:

File Input/Output		
`-I`	`--input-list-file=FILE`	Specifies a file containing a list of input SDFs (one per line).
`-o`	`--output=DIR`	Specifies the directory that will contain the split output bases (must be empty if present).
	`SDF+`	Specifies an input SDF. May be specified 0 or more times.

Utility		
	`--allow-duplicate-names`	Set to disable duplicate name detection. Use this if you need to use less memory and you are certain there are no duplicate names in the input.
`-h`	`--help`	Prints help on command-line flag usage.
	`--in-memory`	Process in memory instead of from disk. (Faster but requires more RAM).
`-n`	`--num-sequences`	Specifies the number of sequences allowed in each SDF. Generally, this command is used to split up read data sets of considerable size.

Usage:

Use the `sdfsplit` command to break up very large read data sets into manageable chunks for processing. Use `-o` to specify the top level output directory and specify the input directories as a space separated list of paths. The subdirectories are constructed underneath the top level output directory.

The `-n` flag specifies the sequence count in each of the newly created SDF directories. Select the value here to match the RAM availability on the server node used for mapping and alignment.

The `-I`, `--input-list-file` flag allows aggregation of multiple SDF directories into one large data set, which can then be split into chunks of appropriate size for the machine configuration available.

For example, an organization has been using server nodes with 48 GB of RAM. They split up the read data sets to optimize processing in this environment. Next year, they buy new server nodes with 96 GB of RAM. They want to rerun the reads against a new reference, so they use all of the existing read data set SDF directories as input into *sdfsplit* and create new SDF directories with more reads in each.

Several RTG commands, like `map`, now have `--start-read` and `--end-read` flag options that may be preferable to using `sdfsplit` in most situations.

See also:

format, *cg2sdf*, *sdf2fasta*, *sdfstats*, *sdfsplit*

2.11.7 sdfsubset

Synopsis:

Extracts a specified subset of sequences from one SDF and outputs them to another SDF.

Syntax:

Individual specification of sequence ids:

```
$ rtg sdfsubset [OPTION]... -i SDF -o SDF STRING+
```

File list specification of sequence ids:

```
$ rtg sdfsubset [OPTION]... -i SDF -o SDF -I FILE
```

Example:

```
$ rtg sdfsubset -i reads -o subset_reads 10 20 30 40 50
```

Parameters:

File Input/Output		
-i	--input=SDF	Specifies the input SDF.
-o	--output=SDF	The name of the output SDF.

Filtering		
	--end-id=INT	Only output sequences with sequence id less than the given number. (Sequence ids start at 0).
	--start-id=INT	Only output sequences with sequence id greater than or equal to the given number. (Sequence ids start at 0).
-I	--id-file=FILE	Name of a file containing a list of sequences to extract, one per line.
	--names	Interpret any specified sequence as names instead of numeric sequence ids.
	STRING+	Specifies the sequence id, or sequence name if the names flag is set to extract from the input SDF. May be specified 0 or more times.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

Use this command to obtain a subset of sequences from an SDF. Either specify the subset on the command line as a list of space-separated sequence ids or using the `--id-file` parameter to specify a file containing a list of sequence ids, one per line. Sequence ids start from zero and are the same as the ids that `map` uses by default in the `QNAME` field of its BAM files.

For example:

```
$ rtg sdfsubset -i reads -o subset_reads 10 20 30 40 50
```

This will produce an SDF called subset_reads with sequences 10, 20, 30, 40 and 50 from the original SDF contained in it.

See also:

sdfsubseq, *sdfstats*

2.11.8 sdfsubseq

Synopsis:

Prints a subsequence of a given sequence in an SDF.

Syntax:

Print sequences from sequence names:

```
$ rtg sdfsubseq [OPTION]... -i FILE STRING+
```

Print sequences from sequence ids:

```
$ rtg sdfsubseq [OPTION]... -i FILE -I STRING+
```

Example:

```
$ rtg sdfsubseq -i reads -I 0:1+100
```

Parameters:

File Input/Output		
-i	--input=FILE	Specifies the input SDF.

Filtering		
-I	--sequence-id	If set, use sequence id instead of sequence name in region (0-based)
	REGION+	The range to display. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>. Must be specified 1 or more times.

Utility		
-f	--fasta	Set to output in FASTA format.
-q	--fastq	Set to output in FASTQ format.
-h	--help	Prints help on command-line flag usage.
-r	--reverse-complement	Set to output in reverse complement.

Usage:

Prints out the nucleotides or amino acids of specified regions in a set of sequences.

For example:

```
$ rtg sdfsubseq --input reads --sequence-id 0:1+20
AGGCGTCTGCAGCCGACGCG
```

See also:

sdfsubset, *sdfstats*

2.11.9 sam2bam

Synopsis:

Convert coordinate sorted SAM/BAM format files to a BAM format file with index.

Syntax:

```
$ rtg sam2bam [OPTION]... -o FILE FILE+
```

Example:

```
$ rtg sam2bam -o alignments.bam alignments.sam.gz
```

Parameters:

File Input/Output		
-o	--output=FILE	Name for output BAM file.
	FILE+	SAM/BAM format files containing mapped reads. Must be specified 1 or more times.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

Use sam2bam to convert SAM/BAM files containing mapped reads to BAM format. This command will preserve alignment calibration information when present, by producing a calibration file alongside the output BAM file.

If additional filtering of alignments is required, use sammerge.

See also:

map, *cgmap*, *sammerge*

2.11.10 sammerge

Synopsis:

Merge and filter coordinate sorted SAM/BAM files into one SAM/BAM output.

Syntax:

Multi-file input specified from command line:

```
$ rtg sammerge [OPTION]... FILE+
```

Multi-file input specified in a text file:

```
$ rtg sammerge [OPTION]... -I FILE
```

Example:

```
$ rtg sammerge alignments1.bam alignments2.bam -o alignments.bam
```

Parameters:

File Input/Output		
	--bed-regions=FILE	If set, only read SAM records that overlap the ranges contained in the specified BED file.
-I	--input-list-file=FILE	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
-o	--output=FILE	Name for output SAM/BAM file. Use '-' to write to standard output.
	--region=REGION	If set, only process SAM records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
-t	--template=SDF	SDF containing the reference genome to use when decoding CRAM input.
	FILE+	SAM/BAM format files containing coordinate-sorted reads. May be specified 0 or more times.

Sensitivity Tuning		
	`--exclude-duplicates`	Exclude all SAM records flagged as a PCR or optical duplicate.
	`--exclude-mated`	Exclude all mated SAM records.
	`--exclude-unmapped`	Exclude all unmapped SAM records.
	`--exclude-unmated`	Exclude all unmated SAM records.
	`--exclude-unplaced`	Exclude all SAM records with no alignment position.
`-F`	`--filter-flags=INT`	Decimal mask indicating SAM FLAG bits that must not be set for the record.
	`--invert`	If set, invert the result of flag and attribute based filter criteria.
`-m`	`--max-as-mated=INT`	If set, ignore mated SAM records with an alignment score (AS attribute) that exceeds this value.
`-u`	`--max-as-unmated=INT`	If set, ignore unmated SAM records with an alignment score (AS attribute) that exceeds this value.
`-c`	`--max-hits=INT`	If set, ignore SAM records with an alignment count that exceeds this value.
	`--min-mapq=INT`	If set, ignore SAM records with MAPQ less than this value.
	`--min-read-length=INT`	If set, ignore SAM reads with read length less than this value.
	`--remove-duplicates`	Detect and remove duplicate reads based on mapping position.
`-f`	`--require-flags=INT`	Decimal mask indicating SAM FLAG bits that must be set for the record.
`-r`	`--select-read-group=STRING`	Select only SAM records with this read group ID. May be specified 0 or more times, or as a comma separated list.

Utility		
`-h`	`--help`	Print help on command-line flag usage.
	`--legacy-cigars`	If set, produce legacy cigars (using M rather than X or =) in output.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--no-header`	Prevent SAM/BAM header from being written.
	`--seed=INT`	Seed used during subsampling.
	`--subsample=FLOAT`	If set, subsample the input to retain this fraction of reads.
`-T`	`--threads=INT`	Number of threads (Default is the number of available cores)

Usage:

Use this command to merge multiple sorted SAM/BAM files into one sorted SAM/BAM file. It can also be used to produce a filtered set of SAM records based on the tuning criteria. If the extension of the given output file name is `.bam` the output will be in BAM format instead of SAM format.

When operating on RTG BAM files that have associated calibration files present, `sammerge` will produce a calibration file alongside the output BAM. However, note that when using filtering options that reject or include alignment records according to some criterion, the merged calibration may not accurately reflect the contents of the output BAM. In this case, a warning is issued, and you should consider running `calibrate` separately on the newly created BAM file.

See also:

map, *cgmap*, *samstats*

2.11.11 samstats

Synopsis:

Print alignment statistics from the contents of the output SAM/BAM file.

Syntax:

```
$ rtg samstats [OPTION]... -t SDF FILE+
```

Example:

```
$ rtg samstats -t genome -i alignments.bam
```

Parameters:

File Input/Output		
-I	--input-list-file=FILE	Specifies a file containing a list of SAM/BAM format files (one per line) containing mapped reads.
-r	--reads=SDF	Specifies the SDF containing the reads.
-t	--template=SDF	Specifies the reference genome SDF.
	FILE+	Specifies a SAM/BAM result file (must contain read-ids not read names). May be specified 0 or more times.

Reporting		
	--consensus	Set to record consensus data. Requires roughly 5 fold reference genome length of RAM.
-D	--distributions	Set to display distributions of insert sizes, alignment scores and read hits.
	--per-file	Set to output per-file statistics as well as the summary of all SAM/BAM files.
	--validate	Set to validate mapping of read to reference. Tests matching of bases according to CIGAR format.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

Use the `samstats` command to display information about a SAM/BAM file and the mapping run that created it. When used without the original reads, `samstats` reports on the file contents: total records, number unmapped and percentage accuracy of alignments compared to the reference.

When the original reads are included with the `-r` flag, the command reports more information about this particular SAM/BAM file in the context of the entire read data set. This choice reports: reads reported one or more times in the SAM/BAM file compared to the total number of reads in the SDF, the number of reads mapped at a single location (i.e. uniquely), the maximum number of records reported for a read set by the `--max-top-results` flag in the `map` command, and counts of the number of reads mapped at each top results level up to the maximum allowed.

For paired-end reads, the command additionally reports a distribution for the direction of the mate pairs: `FF` (forward-forward), `RF` (reverse-forward), `FR` (forward-reverse), and `RR` (reverse-reverse).

Add the `--consensus` flag to report the coverage depth across the entire alignment file and a consensus percentage. Consensus measures percentage agreement of alignments at base pair locations across the reference.

Set the `--distributions` flag to report summary detail on the number of reads mapped by alignment score (`AS` field). For mated paired-end reads, a distribution of insert size is reported.

Set the `--validate` flag to force the reporting of problems in the alignments file.

See also:

sdfstats

2.11.12 samrename

Synopsis:

Replace read identifiers (`QNAME` field) in a SAM/BAM file generated by the RTG `map` command with the sequence identifiers from the original sequence file.

Syntax:

```
$ rtg samrename [OPTION]... -i SDF FILE
```

Example:

```
$ rtg samrename -i reads alignments.bam
```

Parameters:

File Input/Output		
-i	--input=SDF	Specifies the SDF containing the reads in the SAM/BAM file.
-o	--output=FILE	Specifies the name for the output SAM/BAM file.
	FILE	Specifies the input SAM/BAM file.

Filtering		
	--end-read=INT	Set the exclusive upper bound of the read id set to rename.
	--start-read=INT	Set the inclusive lower bound of the read id set to rename.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.

Usage:

By default the `map` and `cgmap` commands will populate the SAM/BAM output files with internal numeric read identifiers rather than the original read names. The `samrename` command replaces those internal read identifiers with the original read names. If the output file is not specified, the command creates the new file in the same directory as the input file, adding `_rename` to the file name. For example, `alignments.bam` becomes `alignments_rename.bam`.

See also:

map, *samstats*

2.11.13 mapxrename

Synopsis:

Replaces read identifiers (`read-id` field) in a `mapx` output file generated by the RTG `mapx` command with the sequence identifiers from the original sequence file.

Syntax:

```
$ rtg mapxrename [OPTION]... -i SDF FILE
```

Example:

```
$ rtg mapxrename -i human_protein_reads mapx_out.txt.gz
```

Parameters:

File Input/Output		
-i	--input=SDF	SDF for the reads in the mapx file.
-o	--output=FILE	Renamed output mapx file.
	FILE	Input mapx file.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

By default the `mapx` command will populate the output files with internal numeric read identifiers rather than the original read names. The `mapxrename` command replaces those internal read identifiers with the original read names. If the output file is not specified, the command creates the new file in the same directory as the input file,

adding `_rename` to the file name. For example, `alignments.tsv.gz` becomes `alignments_rename.tsv.gz`.

See also:

mapx

2.11.14 chrstats

Synopsis:

The `chrstats` command checks chromosome coverage levels based on calibration files and produces warnings if levels depart from expected coverage levels.

Syntax:

```
$ rtg chrstats [OPTION]... -t SDF FILE+
```

Example:

Check all samples using sex information from pedigree:

```
$ rtg chrstats -t genome_ref --pedigree ceu_trio.ped trio_map/alignments.bam

sample   specified  consistent  possible  coverage
-----------------------------------------------------
NA12892  FEMALE     true                  56.00
NA12891  MALE       true                  51.75
NA12878  FEMALE     true                  58.11
```

Check a single sample without pedigree:

```
$ rtg chrstats -t genome_ref --sample NA12878 --sex=female \
  NA12878_map/alignments.bam

sample   specified  consistent  possible  coverage
-------------------------------------------------------------------------------
↪----
NA12878  MALE       false       FEMALE       58.34   (2 of 25 sequences have␣
↪unexpected coverage level)
```

Parameters:

File Input/Output		
`-I`	`--input-list-file=FILE`	File containing a list of SAM/BAM format files (1 per line) containing mapped reads.
`-t`	`--template=SDF`	SDF containing the reference genome.
	`FILE+`	alignment files to process. Must be specified 1 or more times

Sensitivity Tuning		
	`--output-pedigree=FILE`	output best guest of per-sample sex information to PED file
`-s`	`--sample=STRING`	the name of the sample to check (required when checking single sample from multiple samples alignments)
	`--sex=SEX`	sex setting that the individual was mapped as (when not using pedigree). Allowed values are [male, female, either] (Default is either)
	`--pedigree=FILE`	Genome relationships PED file containing sample sex information.
	`--sex-z-threshold=NUM`	The z-score deviation threshold for sex chromosome consistency (Default is 5.0)
	`--z-threshold=NUM`	The z-score deviation threshold for chromosome consistency (Default is 10.0)

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

Given a set of alignments which represent genomic mapping for one or more samples, the `chrstats` command examines chromosomal coverage levels and checks their expected levels with respect to each other. This can be used to indicate gross chromosomal abnormalities, or cases where the sample sex does not match expected (e.g. due to sample mislabelling, incorrect pedigree sex information, etc)

To ensure correct identification of expected ploidy on autosomes and sex chromosomes it is necessary to specify a template containing an appropriate `reference.txt` file. See *RTG reference file format* for more information on `reference.txt` files.

While it is best to give the template used during mapping, for checking third-party outputs any template containing the same chromosome names and an appropriate `reference.txt` file will work. Note that the input alignment files must have calibration information, as automatically produced during mapping by the `map` or `cgmap` commands, or explicitly created by the `calibrate` command.

This command can be used with the results of either whole genome or exome sequencing, although the latter requires that mapping (or subsequent calibration) employed the `--bed-regions` flag.

See also:

map, *cgmap*, *calibrate*

2.11.15 mendelian

Synopsis:

The `mendelian` command checks a multi-sample VCF file for variant calls which do not follow Mendelian inheritance, and compute aggregate sample concordance.

Syntax:

```
$ rtg mendelian [OPTION]... -i FILE -t SDF
```

Example:

```
$ rtg mendelian -i family.vcf.gz -t genome_ref
```

Parameters:

File Input/Output		
-i	--input=FILE	VCF file containing multi-sample variant calls. Use '-' to read from standard input.
-o	--output=FILE	If set, output annotated calls to this VCF file. Use '-' to write to standard output.
	--output-consistent=FILE	If set, output only consistent calls to this VCF file.
	--output-inconsistent=FILE	If set, output only non-Mendelian calls to this VCF file.
-t	--template=SDF	SDF containing the reference genome.

Sensitivity Tuning		
	--all-records	Use all records, regardless of filters (Default is to only process records where FILTER is `.` or `PASS`)
-l	--lenient	Allow homozygous diploid calls in place of haploid calls and assume missing values are equal to the reference.
	--min-concordance=FLOAT	Percentage concordance required for consistent parentage (Default is 99.0)
	--pedigree=FILE	Genome relationships PED file (Default is to extract pedigree information from VCF header fields)

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.

Usage:

Given a multi-sample VCF file for a nuclear family with a defined pedigree, the `mendelian` command examines the variant calls and outputs the number of violations of Mendelian inheritance. If the `--output-inconsistent` parameter is set, all detected violations are written into an output VCF file. As such, this command may be regarded as a VCF filter, outputting those variant calls needing a non-Mendelian explanation. Such calls may be the consequence of sequencing error, calling on low-coverage, or genuine novel variants in one or more individuals.

Pedigree information regarding the relationships between samples and the sex of each sample is extracted from the VCF headers automatically created by the RTG pedigree-aware variant calling commands. If this pedigree information is absent from the VCF header or is incorrect, a pedigree file can be explicitly supplied with the `--pedigree` flag.

To ensure correct behavior when dealing with sex chromosomes it is necessary to specify a sex-aware reference and ensure the sex of each sample is supplied as part of the pedigree information. While it is best to give the reference SDF used in the creation of the VCF, for checking third-party outputs any reference SDF containing the same chromosome names and an appropriate `reference.txt` file will work. For more information, see *RTG reference file format*. Variants calls where the call ploidy does not match what is expected are annotated in the output VCF with an `MCP` FORMAT annotation.

Particularly when evaluating VCF files that have been produced by third party tools or when the VCF is the result of combining independent per-sample calling, it is common to end up with situations where calls are not available for every member of the family. Under normal circumstances `mendelian` will attempt to determine Mendelian consistency on the basis of the values that have been provided. Records where the presence of missing values makes the Mendelian consistency undecidable contain `MCU` INFO annotations in the annotated output VCF. The following examples illustrate some consistent, undecidable, and inconsistent calls in the presence of missing values:

```
CHROM  FATHER_GT  MOTHER_GT  SON_GT  STATUS
chrX   .          0/1        1       OK
chr1   ./.        1/1        1/2     MCU
chr1   ./.        1/1        2/2     MCV
```

Since the number of calls where one sample is missing can be quite high, an alternative option is to treat missing values as equal to the reference by using the `--lenient` parameter. Note that while this approach will be correct in most cases, it will give inaccurate results where the calling between different samples has reported the variant in an equivalent but slightly different position or representation (e.g. positioning of indels within homopolymer regions, differences of representation such as splitting MNPs into multiple SNPs etc).

The `mendelian` command computes overall concordance between related samples to assist detecting cases where pedigree has been incorrectly recorded or samples have been mislabelled. For each child in the pedigree, pairwise concordance is computed with respect to each parent by identifying diploid calls where the parent does not contain either allele called in the child. Low pairwise concordance with a single parent may indicate that the parent is the source of the problem, whereas low pairwise concordance with both parents may indicate that the child is the source of the problem. A stricter three-way concordance is also recorded.

By default, only VCF records with the `FILTER` field set to PASS or missing are processed. All variant records can be examined by specifying the `--all-records` parameter.

See also:

family, *population*, *vcfstats*

2.11.16 vcfstats

Synopsis:

Display simple statistics about the contents of a set of VCF files.

Syntax:

```
$ rtg vcfstats [OPTION]... FILE+
```

Example:

```
$ rtg vcfstats /data/human/wgs/NA19240/snp_chr5.vcf.gz

Location                       : /data/human/wgs/NA19240/snp_chr5.vcf.gz
Passed Filters                 : 283144
Failed Filters                 : 83568
SNPs                           : 241595
MNPs                           : 5654
Insertions                     : 15424
Deletions                      : 14667
Indels                         : 1477
Unchanged                      : 4327
SNP Transitions/Transversions : 1.93 (210572/108835)
Total Het/Hom ratio            : 2.13 (189645/89172)
SNP Het/Hom ratio              : 2.12 (164111/77484)
MNP Het/Hom ratio              : 3.72 (4457/1197)
Insertion Het/Hom ratio        : 1.69 (9695/5729)
Deletion Het/Hom ratio         : 2.33 (10263/4404)
Indel Het/Hom ratio            : 3.13 (1119/358)
Insertion/Deletion ratio       : 1.05 (15424/14667)
Indel/SNP+MNP ratio            : 0.13 (31568/247249)
```

Parameters:

File Input/Output		
	`--known`	Set to only calculate statistics for known variants.
	`--novel`	Set to only calculate statistics for novel variants.
	`--sample=FILE`	Set to only calculate statistics for the specified sample. (Default is to include all samples). May be specified 0 or more times.
	FILE+	VCF file from which to derive statistics. Use '-' to read from standard input. Must be specified 1 or more times.

Reporting		
	`--allele-lengths`	Set to output variant length histogram.

Utility		
`-h`	`--help`	Prints help on command-line flag usage.

Usage:

Use the `vcfstats` command to display summary statistics for a set of VCF files. If a VCF file contains multiple sample columns, the statistics for each sample are shown individually.

When determining the categorization of a REF to ALT transformation, some normalization is carried out to ignore same as reference bases at the start and end of the alleles. Thus the following REF to ALT transformations are categorized as `SNPs`:

```
A    -> G      (simple case)
ATGC -> ATGG   (leading bases match)
ATGC -> ACGC   (leading and trailing bases match)
```

Cases where multiple bases change, but the lengths of the two alleles do not are considered to be `MNPs`:

```
ATGC -> TTGG   (two bases change)
ATGC -> GTCT   (three bases change)
```

Cases where there is pure addition or removal of bases are classified as `Insertions` or `Deletions` respectively:

```
A     -> AT      (one base insertion)
ATT   -> ATTTT   (two base insertion)
AT    -> A       (one base deletion)
ATTTT -> ATT     (two base deletion)
```

The remaining case is there there is a length change between the REF and ALT, but it is not pure. These are called `Indels`:

```
ATT   -> CTTT    (one base changed, one base inserted)
CTTT  -> ATT     (one base changed, one base deleted)
```

In the per-sample summary output of `vcfstats`, each genotype is classified as a whole into one of the above categories, preferring the more complex of the transformations when ploidy is greater than one.

When computing the per-sample variant length histograms, note that the histograms are incremented for each called allele (thus a diploid homozygous call will increment the appropriate cell by two), and the length of an indel is taken as the change in length rather than the overall length.

See also:

snp, *family*, *somatic*, *vcffilter*, *vcfmerge*, *vcfsubset*

2.11.17 vcfmerge

Synopsis:

Combines the contents of two or more VCF files. The `vcfmerge` command can concatenate the outputs of per-chromosome variant detection runs to create a complete genome VCF file, and also merge VCF outputs from multiple samples to form a multi-sample VCF file.

Syntax:

```
$ rtg vcfmerge [OPTION]... -o FILE FILE+
```

Example:

```
$ rtg vcfmerge -o merged.vcf.gz snp1.vcf.gz snp2.vcf.gz
```

Parameters:

File Input/Output		
`-I`	`--input-list-file=FILE`	File containing a list of VCF format files (1 per line) to be merged.
`-o`	`--output=FILE`	Output VCF file. Use '-' to write to standard output.
	`FILE+`	Input VCF files to merge. May be specified 0 or more times.

Utility		
`-a`	`--add-header=STRING|FILE`	File containing VCF header lines to add, or a literal header line. May be specified 0 or more times.
`-f`	`--force-merge=STRING`	Allow merging of specified header ID even when descriptions do not match. May be specified 0 or more times.
`-F`	`--force-merge-all`	Attempt merging of all non-matching header declarations.
`-h`	`--help`	Print help on command-line flag usage.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--preserve-formats`	If set, variants with different ALTs and unmergeable FORMAT fields will be kept unmerged (Default is to remove those FORMAT fields so the variants can be combined)
	`--stats`	Output statistics for the merged VCF file.

Usage:

The `vcfmerge` command takes a list of VCF files and outputs to a single VCF file. The input files must have consistent header lines, although similar header lines can be forced to merge using the `--force-merge` param-

eter. Each VCF file must be block compressed and have a corresponding tabix index file, which is the default for outputs from RTG variant detection tools, but may also be created from an existing VCF file using the RTG `bgzip` and `index` commands.

There are two primary usage scenarios for the `vcfmerge` command. The first is to combine input VCFs corresponding to different genomic regions (for example, if variant calling was carried out for each chromosome independently on different nodes of a compute cluster). The second scenario is when combining VCFs containing variant calls for different samples (e.g. combining calls made for separate cohorts into a single VCF). If the input VCFs contain multiple calls at the same position for the same sample, a warning is issued and only the first is kept.

When multiple records occur at the same position and the length on the reference is the same, the records will be merged into a single record. If the merge results in a change in the set of ALT alleles, any VCF `FORMAT` fields declared to be of type `A`, `G`, or `R` will be set to the missing value (`.`), as they cannot be meaningfully updated. Similarly, if multiple input records with the same position and length on the reference contain information for the same sample, only that information from the first record will be retained. The `--preserve-formats` flag prevents this loss of information by refusing to merge the records when these conditions occur (separate records will be output).

The `--add-header` option allows inserting arbitrary VCF header lines into the output VCF. For more information, see *vcfannotate*.

See also:

snp, *family*, *population*, *somatic*, *vcffilter*, *vcfannotate*, *vcfsubset*, *bgzip*, *index*

2.11.18 vcffilter

Synopsis:

Filters VCF records based on various criteria. When filtering on multiple samples, if any of the specified samples fail the criteria, the record will be filtered. By default filtered records are removed, but see the –fail, –clear-failed-samples, and –fail-samples options for alternatives.

Syntax:

```
$ rtg vcffilter [OPTION]... -i FILE -o FILE
```

Examples:

Keep only records where the sample has depth of coverage at least 5:

```
$ rtg vcffilter -i snps.vcf.gz -o snps_cov5.vcf.gz -d 5
```

Keep only biallelic records:

```
$ rtg vcffilter -i snps.vcf.gz -o snps_biallelic.vcf.gz --max-alleles 2
```

Parameters:

File Input/Output		
	`--all-samples`	Apply sample-specific criteria to all samples contained in the input VCF.
	`--bed-regions=FILE`	If set, only read VCF records that overlap the ranges contained in the specified BED file.
`-i`	`--input=FILE`	VCF file containing variants to be filtered. Use '-' to read from standard input.
`-o`	`--output=FILE`	Output VCF file. Use '-' to write to standard output. This option is required, unless `--javascript` is being used.
	`--region=REGION`	If set, only read VCF records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
	`--sample=STRING`	Apply sample-specific criteria to the named sample contained in the input VCF. May be specified 0 or more times.

Filtering (Record based)		
-w	--density-window=INT	Window within which multiple variants are discarded.
	--exclude-bed=FILE	Discard all variants within the regions in this BED file.
	--exclude-vcf=FILE	Discard all variants that overlap with the ones in this file.
	--include-bed=FILE	Only keep variants within the regions in this BED file.
	--include-vcf=FILE	Only keep variants that overlap with the ones in this file.
-j	--javascript=STRING	Javascript filtering functions for determining whether to keep record. May be either an expression or a file name. May be specified 0 or more times. *See Examples*
-e	--keep-expr=STRING	Records for which this expression evaluates to true will be retained. *See Examples*
-k	--keep-filter=STRING	Only keep variants with this FILTER tag. May be specified 0 or more times, or as a comma separated list.
-K	--keep-info=STRING	Only keep variants with this INFO tag. May be specified 0 or more times, or as a comma separated list.
	--max-alleles=INT	Maximum number of alleles (REF included)
-C	--max-combined-read-depth=INT	Maximum allowed combined read depth.
-Q	--max-quality=FLOAT	Maximum allowed quality.
	--min-alleles=INT	Minimum number of alleles (REF included)
-c	--min-combined-read-depth=INT	Minimum allowed combined read depth.
-q	--min-quality=FLOAT	Minimum allowed quality.
-r	--remove-filter=STRING	Remove variants with this FILTER tag. May be specified 0 or more times, or as a comma separated list.
-R	--remove-info=STRING	Remove variants with this INFO tag. May be specified 0 or more times, or as a comma separated list.
	--remove-overlapping	Remove records that overlap with previous records.

Filtering (Sample based)		
-A	--max-ambiguity-ratio=FLOAT	Maximum allowed ambiguity ratio.
	--max-avr-score=FLOAT	Maximum allowed AVR score.
	--max-denovo-score=FLOAT	Maximum de novo score threshold.
-G	--max-genotype-quality=FLOAT	Maximum allowed genotype quality.
-D	--max-read-depth=INT	Maximum allowed sample read depth.
	--min-avr-score=FLOAT	Minimum allowed AVR score.
	--min-denovo-score=FLOAT	Minimum de novo score threshold.
-g	--min-genotype-quality=FLOAT	Minimum allowed genotype quality.
-d	--min-read-depth=INT	Minimum allowed sample read depth.
	--non-snps-only	Only keep where sample variant is MNP or INDEL.
	--remove-all-same-as-ref	Remove where all samples are same as reference.
	--remove-hom	Remove where sample is homozygous.
	--remove-same-as-ref	Remove where sample is same as reference.
	--snps-only	Only keep where sample variant is a simple SNP.

Reporting		
	--clear-failed-samples	Retain failed records, set the sample GT field to missing.
-f	--fail=STRING	Retain failed records, add the provided label to the FILTER field.
-F	--fail-samples=STRING	Retain failed records, add the provided label to the sample FT field.

Utility		
-a	--add-header=STRING|FILE	File containing VCF header lines to add, or a literal header line. May be specified 0 or more times.
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--no-header	Prevent VCF header from being written.

Usage:

Use `vcffilter` to get a subset of the results from variant calling based on the filtering criteria supplied by the filter flags. Multiple criteria can be specified at once, and advanced processing can be specified via JavaScript scripting.

When filtering on multiple samples, if any of the specified samples fail the criteria, the record will be filtered. The default behavior is for filtered records to be excluded from output altogether, but alternatively the records can be retained but with an additional user-specified VCF `FILTER` status set via `--fail` option, or if sample-specific filtering criteria is being applied, only those samples can be filtered either by setting their `GT` field to missing by using the `--clear-failed-samples` option, or by setting the `FORMAT` `FT` field with a user-specified status via the `--fail-samples` option.

The `--bed-regions` option makes use of tabix indexes to avoid loading VCF records outside the supplied regions, which can give faster filtering performance. If the input VCF is not indexed or being read from standard input, or if records failing filters are to be annotated via the `--fail` option, use the `--include-bed` option instead.

The flags `--min-denovo-score` and `--max-denovo-score` can only be used on a single sample. Records will only be kept if the specified sample is flagged as a *de novo* variant and the score is within the range specified by the flags. It will also only be kept if none of the other samples for the record are also flagged as a *de novo* variant within the specified score range.

The `--add-header` option allows inserting arbitrary VCF header lines into the output VCF. For more information, see *vcfannotate*.

A powerful general-purpose filtering capability has been included that permits the specification of filter criteria as simple JavaScript expressions (`--keep-expr`) or more comprehensive JavaScript processing functions (`--javascript`). Both `--keep-expr` and `--javascript` can take JavaScript on the command line or if a filename is supplied then the script/expression will be read from that file. `--keep-expr` will be applied before `--javascript`, so the `--javascript` record function will not be called for records filtered out by `--keep-expr`.

See also:

For full details of functions available in `--keep-expr` and `--javascript` see *RTG JavaScript filtering API*

Simple filtering by JavaScript expression with `--keep-expr`

The `--keep-expr` flag aims to provide a convenient way to apply some simple (typically one line) filtering expressions which are evaluated in the context of each record. The final expression of the fragment must evaluate to a boolean value. Records which evaluate to `true` will be retained, while `false` will be removed. The value must be of type boolean, simply being truthy/falsy (in the JavaScript sense) will raise an error.

`--keep-expr` examples:

The following expression keeps records where the `NA12878` sample has `GQ` > 30 and the total depth is > 20. JavaScript will auto convert numerical strings when comparing a string with a number, so calls to `parseInt` can be omitted.

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
--keep-expr "'NA12878'.GQ > 30 && INFO.DP > 20"
```

If the field of interest may contain the missing value ('.') or may be entirely missing on a per-record basis, the `has()` function can be used to control whether such records are kept vs filtered. For example, to keep records with depth greater than 20, and remove any without a `DP` annotation:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
--keep-expr "has(INFO.DP) && INFO.DP > 20"
```

Alternatively, to keep records with depth greater than 20, as well as those without a DP annotation:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
--keep-expr "!has(INFO.DP) || INFO.DP > 20"
```

The next example keeps records where all samples have a depth > 10. The standard JavaScript array methods `every` and `some` can be used to apply a condition on every sample column.

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
  --keep-expr "SAMPLES.every(function(s) {return s.DP > 10})"
```

Similarly, the following example retains records where the `FILTER` field is unset, or if set must be either `PASS` or `MED_QUAL`:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
  --keep-expr "FILTER.every(function(f) {return f == 'PASS' || f == 'MED_QUAL'})"
```

Note that multi-valued `INFO` and `FORMAT` fields are not split into sub-values, so in some cases correct filtering may require splitting the values first. For example, to select bi-allelic records with `AF` greater than 0.1, the following simple selection will work:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
  --keep-expr "INFO.AF>=0.1"
```

However, in the presence of multi-allelic records, something like the following is required:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz \
  --keep-expr "INFO.AF.split(',').some(function(af) {return af >= 0.1})"
```

Advanced JavaScript filtering with `--javascript`

The `--javascript` option aims to support more complicated processing than `--keep-expr`. permitting modification of the output VCF, or supporting use cases where the script is tasked to compute and output alternative information in addition to (or instead of) the output VCF. The scripts specified by the user are evaluated once at the start of processing. Two special functions may be defined in a `--javascript` script, which will then be executed in different contexts:

- A function with the name `record` will be executed once for each VCF record. If the `record` function has a return value it must have type boolean. Records which evaluate to `true` will be retained, while `false` will be removed. If the record function has no return value then the record will be retained. The `record` function is applied after any `--keep-expr` expression.
- A function with the name `end` will be called once at the end of processing. This allows reporting of summary statistics collected during the filter process.

This `--javascript` flag may be specified multiple times, they will be evaluated in order, in a shared JavaScript namespace, before VCF processing commences. This permits a use case where an initial JavaScript expression supplies parameter values which will be required by a subsequent JavaScript file.

Example `--javascript` scripts:

To find indels with length greater than 5, save the following to a file named `find-indels.js`:

```
// Finds indels with length > 5
function record() {
  var deltas = ALT.map(function (alt) {
    return Math.abs(alt.length - REF.length);
  });
  return deltas.some(function (delta) {return delta > 5});
}
```

Then perform the filtering via:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz --javascript find-indels.js
```

The following example derives a new FORMAT column containing variant allelic fraction to two decimal places based on the values in the AD and DP FORMAT annotations, for every sample contained in the VCF. Save the following to a file named add-vaf.js:

```
// Derive new VAF FORMAT field for each sample
ensureFormatHeader('##FORMAT=<ID=VAF,Number=1,Type=Float,' +
  'Description="Variant Allelic Fraction">');

function record() {
  SAMPLES.forEach(function(sample) {
    // Take all but the first AD value as numerics
    var altDepths = sample.AD.split(",").slice(1);
    // Find the max
    var maxAltDepth = Math.max.apply(null, altDepths);
    if (maxAltDepth > 0) {
      sample.VAF = (maxAltDepth / sample.DP).toFixed(2);
    }
  });
}
```

Then run the filtering via:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz --javascript add-vaf.js
```

The next example produces a table of binned indel lengths, save the following to a file named indel-lengths.js:

```
// bin breakpoints can be customised by defining your own bins[] in a
// previous -j flag
if (typeof bins == "undefined") {
  var bins = [-10, -5, -3, 0, 4, 6, 11];
}

var counts = [0];
bins.forEach(function () {counts.push(0)});
function record() {
  if (ALT.length == 0) {
    return false;
  }
  var deltas = ALT.map(function (alt) { return alt.length - REF.length; });
  var maxDel = Math.min.apply(null, deltas);
  var maxIns = Math.max.apply(null, deltas);
  var delta = Math.abs(maxDel) > maxIns ? maxDel : maxIns;

  if (delta == 0) {
    return false;
  }
  for (var i = 0; i < bins.length; i++) {
    if (delta < bins[i]) {
      counts[i]++;
      break;
    }
  }
  if (delta > bins[bins.length - 1]) {
    counts[counts.length - 1]++;
  }
  return false;
}
```

```
function end() {
  print("Delta\\tCount");
  for (var i = 0; i < bins.length; i++) {
    print("<" + bins[i] + "\\t" + counts[i]);
  }
  print(">" + bins[bins.length - 1] + "\\t" + counts[counts.length - 1]);
}
```

Then run the filtering via:

```
$ rtg vcffilter -i in.vcf.gz -o out.vcf.gz --javascript indel-lengths.js
```

We could use this same script with adjusted bins and omitting the output of the VCF via:

```
$ rtg vcffilter -i in.vcf.gz -j "var bins = [-20, -10, 0, 20, 20];" \
  -j indel-lengths.js
```

See also:

snp, *family*, *somatic*, *population*, *vcfannotate*, *vcfmerge*, *vcfsubset*

2.11.19 vcfannotate

Synopsis:

Used to add annotations to a VCF file, either to the VCF `ID` field, as a VCF `INFO` sub-field, or as a VCF `FORMAT` sub-field.

Syntax:

```
$ rtg vcfannotate [OPTION]... -b FILE -i FILE -o FILE
```

Example:

```
$ rtg vcfannotate -b dbsnp.bed -i snps.vcf.gz -o snps-dbsnp.vcf.gz
```

Parameters:

File Input/Output		
`-i`	`--input=FILE`	VCF file containing variants to annotate. Use '-' to read from standard input.
`-o`	`--output=FILE`	Output VCF file name. Use '-' to write to standard output.

Reporting		
`-A`	`--annotation=STRING`	Add computed annotation to VCF records. Allowed values are [AC, AN, EP, GQD, IC, LAL, MEANQAD, NAA, PD, QA, QD, RA, SCONT, VAF, VAF1, ZY]. May be specified 0 or more times, or as a comma separated list.
	`--bed-ids=FILE`	Add variant IDs from BED file. May be specified 0 or more times.
	`--bed-info=FILE`	Add INFO annotations from BED file. May be specified 0 or more times.
	`--fill-an-ac`	Add or update the AN and AC INFO fields.
	`--info-description=STRING`	If the BED INFO field is not already declared, use this description in the header (Default is Annotation)
	`--info-id=STRING`	The INFO ID for BED INFO annotations (Default is ANN)
	`--relabel=FILE`	Relabel samples according to `old-name new-name` pairs in specified file.
	`--vcf-ids=FILE`	Add variant IDs from VCF file. May be specified 0 or more times.

Utility		
-a	--add-header=STRING|FILE	File containing VCF header lines to add, or a literal header line. May be specified 0 or more times.
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--no-header	Prevent VCF header from being written.

Usage:

Use `vcfannotate` to add text annotations to variants.

A common use case is to add annotations to only those variants that fall within ranges specified in a BED or VCF file, supplied via `--bed-ids` or `--vcf-ids` respectively. The annotations from the BED file are added as an `INFO` field in the output VCF file. It can also be used to compute or fill in certain additional annotations from the existing content. Note that this annotation method is solely based on the position and span of the variant, ignoring actual alleles and genotypes.

If the `--bed-ids` flag is used, instead of adding the annotation to the `INFO` fields, it is added to the `ID` column of the VCF file instead. If the `--vcf-ids` flag is used, the `ID` column of the input VCF file is used to update the `ID` column of the output VCF file instead.

If the `--fill-an-ac` flag is set, the output VCF will have the `AN` and `AC` info fields (as defined in the VCF 4.1 specification) created or updated.

It is also possible to use `vcfannotate` to insert additional VCF header lines into the VCF header. These are supplied using the `--add-header` flag which may either be a literal VCF header line (useful for adding one or two header lines), or from a file.

```
$ rtg vcfannotate -i in.vcf.gz -o out.vcf.gz \
--add-header "##SAMPLE=<ID=NA24385,Sex=MALE>" \
--add-header "##SAMPLE=<ID=NA24143,Sex=FEMALE>" \
--add-header "##SAMPLE=<ID=NA24149,Sex=MALE>" \
--add-header "##PEDIGREE=<Child=NA24385,Mother=NA24143,Father=NA24149>"
```

or alternatively:

```
$ rtg vcfannotate -i in.vcf.gz -o out.vcf.gz --add-header ped_vcf_headers.txt
```

Care should be taken that the lines being inserted are valid VCF header lines.

If the `--annotation` flag is set, `vcfannotate` attempts to compute the specified annotation(s) and add them as `FORMAT` fields in the corresponding records. Records for which particular annotations cannot be computed, due to a lack of pre-requisite fields, will not be modified.

For a description of the meaning of fields available for annotation, see *Small-variant VCF output file description*. The `SCONT` annotation is a convenience to annotate with all of the contrary evidence annotations: `DCOC`, `DCOF`, `OCOC`, `OCOF`.

See also:

snp, *family*, *somatic*, *population*, *vcffilter*, *vcfsubset*

2.11.20 vcfsubset

Synopsis:

Create a VCF file containing a subset of the original columns.

Syntax:

```
$ rtg vcfsubset [OPTION]... -i FILE -o FILE
```

Example:

```
$ rtg vcfsubset -i snps.vcf.gz -o frequency.vcf.gz --keep-info AF --remove-samples
```

Parameters:

File Input/Output		
-i	--input=FILE	VCF file containing variants to manipulate. Use '-' to read from standard input.
-o	--output=FILE	Output VCF file. Use '-' to write to standard output.

Filtering		
	--keep-filter=STRING	Keep the specified FILTER tag. May be specified 0 or more times, or as a comma separated list.
	--keep-format=STRING	Keep the specified FORMAT field. May be specified 0 or more times, or as a comma separated list.
	--keep-info=STRING	Keep the specified INFO tag. May be specified 0 or more times, or as a comma separated list.
	--keep-sample=STRING	Keep the specified sample. May be specified 0 or more times, or as a comma separated list.
	--remove-filter=STRING	Remove the specified FILTER tag. May be specified 0 or more times, or as a comma separated list.
	--remove-filters	Remove all FILTER tags.
	--remove-format=STRING	Remove the specified FORMAT field. May be specified 0 or more times, or as a comma separated list.
	--remove-ids	Remove the contents of the ID field.
	--remove-info=STRING	Remove the specified INFO tag. May be specified 0 or more times, or as a comma separated list.
	--remove-infos	Remove all INFO tags.
	--remove-qual	Remove the QUAL field.
	--remove-sample=STRING	Remove the specified sample. May be specified 0 or more times, or as a comma separated list.
	--remove-samples	Remove all samples.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--no-header	Prevent VCF header from being written.

Usage:

Use the `vcfsubset` command to produce a smaller copy of an original VCF file containing only the columns and information desired. For example, to produce a VCF containing only the information for one sample from a multiple sample VCF file use the `--keep-sample` flag to specify the sample to keep. The various `--keep` and `--remove` options can either be specified multiple times or with comma separated lists, for example, `--keep-format GT --keep-format DP` is equivalent to `-keep-format GT,DP`.

See also:

snp, *family*, *somatic*, *population*, *vcffilter*, *vcfannotate*

2.11.21 vcfdecompose

Synopsis:

Decomposes complex variants within a VCF file into smaller components.

Syntax:

```
$ rtg vcfdecompose [OPTION]... -i FILE -o FILE
```

Parameters:

File Input/Output		
-i	--input=FILE	VCF file containing variants to decompose. Use '-' to read from standard input.
-o	--output=FILE	Output VCF file name. Use '-' to write to standard output.
-t	--template=SDF	SDF of the reference genome the variants are called against.

Sensitivity Tuning		
	--break-indels	If set, peel as many SNPs off an indel as possible.
	--break-mnps	If set, break MNPs into individual SNPs.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--no-header	Prevent VCF header from being written.

Usage:

The `vcfdecompose` command decomposes and trims variants based on a multiple sequence alignment between the alleles in each VCF record. Only records where every ALT allele is an ordinary allele (i.e. consisting of nucleotides) will undergo decomposition. In addition, if there are redundant same-as-reference bases in the alleles, these will be trimmed off.

The default behaviour is to break the variant at positions where there is at least one base aligned to the reference across all ALT alleles, so the output may contain MNPs or impure indels. If desired, MNPs can be split into individual SNPs via `--break-mnps`. Similarly, impure indels can be split into a combination of SNPs and pure indels via `--break-indels`.

Although decomposed variants carry through the original `INFO` and `FORMAT` annotations, the decomposition may mean that some annotations are no longer semantically correct. In particular, any VCF `FORMAT` fields declared to be of type `A`, `G`, or `R` will no longer be valid if the set of alleles has changed.

Note that the reference genome is an optional parameter. When variants are decomposed and trimmed, the resulting variant may require a padding base to be added, as required by the VCF specification. The VCF specification suggests that the padding base should be the base before the variant (i.e. padding on the left), but sometimes this requires knowledge of reference bases not present in the original record. When the reference genome is supplied, `vcfdecompose` will ensure that any padding bases are added on the left of the variant. If the reference genome is not supplied, padding bases may sometimes be on the right hand side of the variant. For example:

```
1  20  .  GCGCGCGCGCG  TTTGCGCGCTTGCGCGTTT  .  PASS  .              GT  1/0
```

will decompose without a reference genome as:

```
1  20  .  G           TTTG                 .  PASS  ORP=20;ORL=11  GT  1/0
1  25  .  C           CTT                  .  PASS  ORP=20;ORL=11  GT  1/0
```

and with a reference genome (where the reference base at position 19 can be determined to be a `T`) as:

```
1  19  .  T           TTTT                 .  PASS  ORP=20;ORL=11  GT  1/0
1  25  .  C           CTT                  .  PASS  ORP=20;ORL=11  GT  1/0
```

The variants that are left vs right-padded are equivalent and identified as such by haplotype-aware comparison tools such as `vcfeval`.

See also:

vcffilter, *vcfeval*

2.11.22 vcfeval

Synopsis:

Evaluates called variants for agreement with a baseline variant set irrespective of representational differences. Outputs a weighted ROC file which can be viewed with `rtg rocplot` and VCF files containing false positives

(called variants not matched in the baseline), false negatives (baseline variants not matched in the call set), and true positives (variants that match between the baseline and calls).

The baseline variants might be the variants that were used to generate a synthetic simulated sample (such as via `popsim`, `samplesim`, etc), a gold-standard VCF corresponding to a reference sample such as NA12878, or simply an alternative call-set being used as a basis for comparison.

Syntax:

```
$ rtg vcfeval [OPTION]... -b FILE -c FILE -o DIR -t SDF
```

Example:

```
$ rtg vcfeval -b goldstandard.vcf.gz -c snps.vcf.gz -t HUMAN_reference \
  --sample daughter -f AVR -o eval
```

Parameters:

File Input/Output		
`-b`	`--baseline=FILE`	VCF file containing baseline variants.
	`--bed-regions=FILE`	If set, only read VCF records that overlap the ranges contained in the specified BED file.
`-c`	`--calls=FILE`	VCF file containing called variants.
`-e`	`--evaluation-regions=FILE`	If set, evaluate within regions contained in the supplied BED file, allowing transborder matches. To be used for truth-set high-confidence regions or other regions of interest where region boundary effects should be minimized.
`-o`	`--output=DIR`	Directory for output.
	`--region=REGION`	If set, only read VCF records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>
`-t`	`--template=SDF`	SDF of the reference genome the variants are called against.

Filtering		
	`--all-records`	Use all records regardless of FILTER status (Default is to only process records where FILTER is `.` or `PASS`)
	`--decompose`	Decompose complex variants into smaller constituents to allow partial credit.
	`--ref-overlap`	Allow alleles to overlap where bases of either allele are same-as-ref (Default is to only allow VCF anchor base overlap)
	`--sample=STRING`	The name of the sample to select. Use <baseline_sample>,<calls_sample> to select different sample names for baseline and calls. (Required when using multi-sample VCF files)
	`--squash-ploidy`	Treat heterozygous genotypes as homozygous ALT in both baseline and calls, to allow matches that ignore zygosity differences.

Reporting		
`-m`	`--output-mode=STRING`	Output reporting mode. Allowed values are [split, annotate, combine, ga4gh, roc-only] (Default is split)
`-O`	`--sort-order=STRING`	The order in which to sort the ROC scores so that `good` scores come before `bad` scores. Allowed values are [ascending, descending] (Default is descending)
`-f`	`--vcf-score-field=STRING`	The name of the VCF FORMAT field to use as the ROC score. Also valid are `QUAL`, `INFO.<name>` or `FORMAT.<name>` to select the named VCF FORMAT or INFO field (Default is GQ)

Utility		
-h	--help	Prints help on command-line flag usage.
-Z	--no-gzip	Set this flag to create the output files without compression.
-T	--threads=INT	Specify the number of threads to use in a multi-core processor. (Default is all available cores).

Usage:

The vcfeval command can be used to generate VCF files containing called variants that were in the baseline VCF, called variants that were not in the baseline VCF and baseline variants that were not in the called variants. It also produces ROC curve data files based on a score contained in a VCF field which show the predictive power of that field for the quality of the variant calls.

When developing and validating sequencing pipelines and variant calling algorithms, the comparison of variant call sets is a common problem. The naïve way of computing these numbers is to look at the same reference locations in the baseline (ground truth) and called variant set, and see if genotype calls match at the same position. However, a complication arises due to possible differences in representation for indels between the baseline and the call sets within repeats or homopolymers, and in multiple-nucleotide polymorphisms (MNPs), which encompass several nearby nucleotides and are locally phased. The vcfeval command includes a novel dynamic-programming algorithm for comparing variant call sets that deals with complex call representation discrepancies, and minimizes false positives and negatives across the entire call sets for accurate performance evaluation. A primary advantage of vcfeval (compared to other tools) is that the evaluation does not depend on normalization or decomposition, and so the results of analysis can easily be used to relate to the original variant calls and their annotations.

Note that vcfeval operates at the level of local haplotypes for a sample, so for a diploid genotype, both alleles must match in order to be considered correct. Some of the vcfeval output modes (described below) automatically perform an additional haploid analysis phase to identify variants which may not have a diploid match but which share a common allele (for example, zygosity errors made during calling). If desired, this more lenient haploid comparison can be used at the outset by setting the --squash-ploidy flag (see below).

Note that variants selected for inclusion in a haplotype cannot be permitted to overlap each other (otherwise the question arises of which variant should have priority when determining the resulting haplotype), and any well-formed call-set should not contain these situations in order to avoid such ambiguity. When such cases are encountered by vcfeval, the best non-overlapping result is determined. A special case of overlapping variants is where calls are denoted as partially the same as the reference (for example, a typical heterozygous call). Strictly speaking such variants are an assertion that the relevant haplotype bases must not be altered from the reference and overlap should not be permitted (this is the interpretation that vcfeval employs by default). However, sometimes as a result of using non-haplotype-aware variant calling tools or when using naïve merging of multiple call sets, a more lenient comparison is desired. The --ref-overlap flag will permit such overlapping variants to both match, as long as any overlap only occurs where one variant or other has asserted haplotype bases as being the same as reference.

Common allele matching with --squash-ploidy

When --squash-ploidy is specified, a haploid match is attempted using *each* of the non-reference alleles used in the sample genotype. For example if the baseline and call VCFs each had a record with the same REF and ALT alleles declared, the following GT fields would be considered a match:

```
0/1, 1/1, 1/2   (genotypes match due to the 1 allele)
0/2, 1/2, 2/2   (genotypes match due to the 2 allele)
```

Thus --squash-ploidy matches any case where the baseline and calls share a common allele. This is most often used to run matching that does not penalize for genotyping errors. For example, it is recommended to use this option when matching somatic variant calls, as since somatic variation is usually associated with variable allelic fractions and heterogeneity that mean strict diploid genotype comparisons are not appropriate.

Comparing with a VCF that has no sample column

A common scenario is to match a call set against a baseline which contains no sample column, where the objective is to identify which baseline alleles which have been called. One example of this is to identify whether calls match a database of known high-priority somatic variants such as COSMIC, or to find calls which have been previously seen in a population allele database such as ExAC. Ordinarily `vcfeval` requires the input VCFs to contain a sample column containing a genotype in the GT field, however, it is possible to specify a special sample name of 'ALT' in order to indicate that the the genotypes for comparison should be derived from the ALT alleles of the record. This can be specified independently for baseline and calls, for example:

```
$ rtg vcfeval -t build37.sdf -b cosmic.vcf.gz -c tumor-calls.vcf.gz \
--squash-ploidy --sample ALT,tumor -o tumor-vs-cosmic
```

Which would perform a haploid matching of the GT of the called sample 'tumor' against all possible haploid genotypes in the COSMIC VCF. The resulting true positives file contains all the calls containing an allele present in the COSMIC VCF.

Note: It is also possible to run a diploid comparison by omitting `--squash-ploidy`, but this is not usually required, and is computationally more intensive since there may be many more possible diploid genotypes to explore, particularly if the ALT VCF contains many multiallelic records.)

Evaluation with respect to regions

When evaluating exome variant calls, it may be useful to restrict analysis only to exome target regions. In this case, supply a BED file containing the list of regions to restrict analysis to via the `--bed-regions` flag. For a quick way to restrict analysis only to a single region, the `--region` flag is also accepted. Note that when restricting analysis to regions, there may be variants which can not be correctly evaluated near the borders of each analysis region, if determination of equivalence would require inclusion of variants outside of the region. For this reason, it is recommended that such regions be relatively inclusive.

When matching against gold standard truth sets which have an accompanying high-confidence regions BED file, the flag `--evaluation-regions` should be used instead of `--bed-regions`, as it has special matching semantics that aims to reduce comparison region boundary effects. When this comparison method is used, call variants which match a baseline variant are only considered a true positive if the baseline variant is inside the high confidence regions, and call variants are only considered false positive if they fall inside the high confidence regions.

vcfeval outputs

The primary outputs of `vcfeval` are VCF files indicating which variants matched between the baseline and the calls VCF, and data files containing information used to generate ROC curves with the `rocplot` command (or via spreadsheet). `vcfeval` supports different VCF output modes which can be selected with the `--output-mode` flag according to the type of analysis workflow desired. The following modes are available:

Split (`--output-mode=split`)

This output mode is the default, and produces separate VCF files for each of the match categories. The individual VCF records in these files are not altered in any way, preserving all annotations present in the input files.

- `tp.vcf` – contains those variants from the *calls* VCF which agree with variants in the baseline VCF
- `tp-baseline.vcf` – contains those variants from the *baseline* VCF which agree with variants in the calls VCF. Thus, the variants in `tp.vcf` and `tp-baseline.vcf` are equivalent. This file can be used to successively refine a highly sensitive baseline variant set to produce a consensus from several call sets.
- `fp.vcf` – contains variants from the *calls* VCF which do not agree with baseline variants.

- `fn.vcf` – contains variants from the *baseline* VCF which were not correctly called.

This mode performs a single pass comparison, either in diploid mode (the default), or haploid mode (if `--squash-ploidy` has been set). The separate output files produced by this mode allow the use of `vcfeval` as an advanced haplotype-aware VCF intersection tool.

Annotate (`--output-mode=annotate`)

This output mode does not split the input VCFs by match status, but instead adds `INFO` annotations containing the match status of each record:

- `calls.vcf` – contains variants from the *calls* VCF, augmented with match status annotations.
- `baseline.vcf` – contains variants from the *baseline* VCF, augmented with match status annotations.

This output mode automatically performs two comparison passes, the first finds diploid matches (assigned a match status of `TP`), and a second pass that applies a haploid mode to the false positives and false negatives in order to find calls (such as zygosity errors) that contain a common allele. This second category of match are annotated with status `FN_CA` or `FP_CA` in the output VCFs, and those calls which do not have any match are assigned status `FN` or `FP`. A status value of `IGN` indicates a VCF record which was ignored (for example, due to having a non-PASS filter status, representing a structural variant, or otherwise containing a non-variant genotype). A status of `OUT` indicates a VCF record which does not contain a match status due to falling outside the evaluation regions when `--evaluation-regions` is being used.

Combine (–output-mode=combine)

This output mode provides an easy way to view the baseline and call variants in a single two-sample VCF.

- `output.vcf` – contains variants from both the *baseline* and *calls* VCFs, augmented with match status annotations. The sample under comparison from each of the input VCFs is extracted as a column in the output. As the VCF records from the baseline and calls typically have very different input annotations which can be difficult to merge, and to keep the output format simple, there is no attempt to preserve any of the original variant annotations.

As with the annotation output mode, this output mode automatically performs two comparison passes to find both diploid matches and haploid (lenient) matches.

ROC-only (–output-mode=roc-only)

This output mode provides a lightweight way to run performance benchmarking, as VCF file output is omitted, and only ROC data files are produced.

All of the output modes produce the following ROC data files:

- `weighted_roc.tsv` – contains ROC data derived from all analyzed call variants, regardless of their representation. Columns include the score field, and standard accuracy metrics such as true positives, false positives, false negatives, precision, sensitivity, and f-measure corresponding to each score threshold.
- `snp_roc.tsv` – contains ROC data derived from only those variants which were represented as SNPs. Since the representation conventions can differ between the baseline and calls, there are some subtleties to be aware of when interpreting metrics such as precision, sensitivity, etc, described below.
- `non_snp_roc.tsv` – contains ROC data derived from those variants which were not represented as SNPs. As above, not all metrics are computed for this file.

Note: In addition, `vcfeval` has an output mode (`--output-mode=ga4gh`) which produces the intermediate evaluation format defined by the GA4GH Benchmarking Team, without additional statistics files. This mode is not generally intended for end users, rather it is used when `vcfeval` is selected as the comparison engine inside the

`hap.py` benchmarking tool see: https://github.com/ga4gh/benchmarking-tools and https://github.com/Illumina/hap.py

Benchmarking comparisons using ROC and precision/sensitivity curves

Multiple ROC data files (from a single or several `vcfeval` runs) can be plotted with the `rocplot` command, which allows output to a PNG or SVG image or analysis in an interactive GUI that provides zooming and visualization of the effects of threshold adjustment. As these files are simple tab-separated-value format, they can also be loaded into a spreadsheet tool or processed with shell scripts.

While ROC curve analysis provides a much more thorough method for examining the performance of a call set with respect to a baseline truth set, for convenience, `vcfeval` also produces a `summary.txt` file which indicates match summary statistics that correspond to two key points on the ROC curve. The first point is where all called variants are included (i.e. no thresholding on a score value); and second point corresponding to a score threshold that maximises the F-measure of the curve. While this latter point is somewhat arbitrary, it represents a balanced tradeoff between precision and sensitivity which is likely to provide a fairer comparison when comparing call sets from different callers.

Note that `vcfeval` reports true positives both counted using the baseline variant representation as well as counted using the call variant representation. When these numbers differ greatly, it indicates a general difference in representational conventions used between the two call sets. Since false negatives can only be measured in terms of the baseline representation, sensitivity is defined as:

$$\text{Sensitivity} = \text{TP}_{\text{baseline}} / (\text{TP}_{\text{baseline}} + \text{FN}).$$

Conversely since false positives can only be measured in terms of the call representation, precision is defined as:

$$\text{Precision} = \text{TP}_{\text{call}} / (\text{TP}_{\text{call}} + \text{FP}).$$

Note: For definitions of the terminology used when evaluating caller accuracy, see: https://en.wikipedia.org/wiki/Receiver_operating_characteristic and https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Benchmarking performance for SNPs versus indels

A common desire is to perform analysis separately for SNPs versus indels. However, it is important to note that due the representation ambiguity problem, it is not always trivial to decide in a global sense whether a variant is a SNP or an indel or other complex variant. A group of variants that may be represented as single SNPs in one call-set may be represented as a single complex variant in another call-set. Consider the following example reference and alternate haplotypes:

```
     12345678901234567
REF: ATCGTAAATAAAATGCA
ALT: ATCGTAAAATAAATGCA
```

One variant caller might represent the haplotypes as the following VCF records:

```
chr1 5 . T TA . . . GT 1/1
chr1 9 . TA T . . . GT 1/1
```

While another variant caller could represent the same haplotypes as:

```
chr1 9 . T A . . . GT 1/1
chr1 10 . A T . . . GT 1/1
```

The decision as to which representation to use is essentially arbitrary, yet one caller has used indels (and no SNPs), and the other has used SNPs (and no indels). For this reason it is certainly a poor idea to attempt to divide baseline and called variants into separate SNP and indel datasets up front and perform evaluation on each set separately, as any variants that use different representation categories will not be matched across the independent comparisons. Any variant-type specific metrics should be computed after matching is carried out on the full variant sets.

Note that when there are different representational conventions between the baseline and calls (or between calls from one variant caller and another), then at some level there is really a semantic difference between a "baseline indel" and a "call-set indel" (or "variant-caller-A indel" and "variant-caller-B indel"), so caution should be applied when making conclusions related to SNP versus indel accuracy.

In the `snp_roc.tsv` and `non_snp_roc.tsv` output files, `vcfeval` notes the number of baseline and call variants of each variant type. When considering benchmarking metrics in the absence of any thresholding with respect to a score field, it is straight-forward to use the previous formulae (i.e. sensitivity is computed using the counts from baseline variants, and precision is computed using the counts from called variants). When computing threshold-specific metrics for ROC data points, the computation is more involved. Since only the call variants contain the score field used to rank variants, the number of (say) TP baseline indels that exceed threshold x is not defined. `vcfeval` computes a scaled count as:

$$\text{TP}_{\text{baseline\_indel}}(x) = \text{TP}_{\text{call\_indel}}(x) \times \text{TP}_{\text{baseline\_indel}}/\text{TP}_{\text{call\_indel}}$$

and thus threshold-specific sensitivity is computed as

$$\text{Sensitivity}_{\text{indel}}(x) = \text{TP}_{\text{baseline\_indel}}(x)/(\text{TP}_{\text{baseline\_indel}} + \text{FN}_{\text{indel}})$$

This scaling ensures that the end point of the variant type specific ROC or precision/sensitivity curve ends at the same point that is obtained when computing metrics without any threshold.

Variant decomposition and benchmarking

In general, it is not necessary to run any variant decomposition and/or normalization on variant call sets prior to evaluation with `vcfeval`, as the haplotype aware matching process can account for representation differences. However, since matching is at the granularity of entire variants, a single long complex call will be categorized as either correct or incorrect, even if part of the call may match. If partial credit in the case of long calls is of interest, `vcfeval` includes an option to internally decompose variants prior to matching, using the `--decompose` flag. This decomposition is applied to both baseline and call variants, and any output VCFs will contain the decomposed representation. External VCF decomposition (with more control over decomposition options) is also available via `rtg vcfdecompose`.

See also:

snp, *popsim*, *samplesim*, *childsim*, *rocplot*, *vcfdecompose*

2.11.23 svdecompose

Synopsis:

Split composite structural variants into a breakend representation.

Syntax:

```
$ rtg svdecompose [OPTION]... -i FILE -o FILE
```

Parameters:

File Input/Output		
`-i`	`--input=FILE`	VCF file containing variants to filter. Use '-' to read from standard input.
`-o`	`--output=FILE`	Output VCF file name. Use '-' to write to standard output.

Utility		
-h	--help	Print help on command-line flag usage.
-Z	--no-gzip	Do not gzip the output.
	--no-header	Prevent VCF header from being written.

Usage:

The svdecompose command is applied to a VCF containing structural variants and converts deletion, insertion, inversion, and tandem duplications with SVTYPE of DEL, INS, INV, and DUP, respectively, into corresponding breakend events with SVTYPE=BND. svdecompose will also decompose sequence-resolved insertions and deletions greater than --min-indel-length into breakend representation. Records of others types are passed through without modification.

This operation can be useful for the purposes of reducing output from various structural variant callers to a common representation to better facilitate comparison with the bndeval command.

For insertions, svdecompose will represent the insertion as breakends between the reference and a "virtual haplotype", where for example, contig "<INS_A>" represents the destination of all insertions made on chromosome A. So if another caller produced a similar insertion event (in position and/or length), the break end versions will also be nearby on the virtual contig. For the following insertions:

```
1  54712   .     T  TTTTTTTTTCTTTCTTTCTTTCTTTCTTTCTTTCTTTCTTTCTTTCTTTC  .  .  .
1  934144  I_7   C  CGGAGGGGAGGGCGCGGAGCGGAGG                            .  .  .
1  934144  I_22  C  CGGAGGGGAGGGCGCGGAGCGGAGGGGAGGGCGCGGAGCGGAGG        .  .  .
```

each insertion gets two breakends like this:

```
1  54712   .     T  T[<INS_1>:54712[   .  .  SVTYPE=BND;CIPOS=0,0
1  54712   .     T  ]<INS_1>:54765]C   .  .  SVTYPE=BND;CIPOS=0,0
1  934144  I_7   C  C[<INS_1>:934144[  .  .  SVTYPE=BND;CIPOS=0,0
1  934144  I_22  C  C[<INS_1>:934144[  .  .  SVTYPE=BND;CIPOS=0,0
1  934144  I_7   C  ]<INS_1>:934168]G  .  .  SVTYPE=BND;CIPOS=0,0
1  934144  I_22  C  ]<INS_1>:934187]G  .  .  SVTYPE=BND;CIPOS=0,0
```

See also:

bndeval

2.11.24 bndeval

Synopsis:

Evaluate called breakends for agreement with a baseline breakend set. Outputs a weighted ROC file which can be viewed with rtg rocplot and VCF files containing false positives (called breakends not matched in the baseline), false negatives (baseline breakends not matched in the call set), and true positives (breakends that match between the baseline and calls).

Syntax:

```
$ rtg bndeval [OPTION]... -b FILE -c FILE -o DIR
```

Parameters:

File Input/Output		
-b	--baseline=FILE	VCF file containing baseline variants.
	--bed-regions=FILE	If set, only read VCF records that overlap the ranges contained in the specified BED file.
-c	--calls=FILE	VCF file containing called variants.
-o	--output=DIR	Directory for output.
	--region=REGION	If set, only read VCF records within the specified range. The format is one of <sequence_name>, <sequence_name>:<start>-<end>, <sequence_name>:<pos>+<length> or <sequence_name>:<pos>~<padding>

Filtering		
	`--all-records`	Use all records regardless of FILTER status (Default is to only process records where FILTER is "." or "PASS")
	`--bidirectional`	If set, allow matches between flipped breakends.
	`--tolerance=INT`	Positional tolerance for breakend matching (Default is 100)

Reporting		
`-m`	`--output-mode=STRING`	Output reporting mode. Allowed values are [split, annotate] (Default is split)
`-O`	`--sort-order=STRING`	The order in which to sort the ROC scores so that "good" scores come before "bad" scores. Allowed values are [ascending, descending] (Default is descending)
`-f`	`--vcf-score-field=STRING`	The name of the VCF field to use as the ROC score. Also valid are "QUAL" or "INFO.<name>" to select the named VCF INFO field (Default is INFO.DP)

Utility		
`-h`	`--help`	Print help on command-line flag usage.
`-Z`	`--no-gzip`	Do not gzip the output.

Usage:

The `bndeval` command operates on VCF files containing breakends such as those produced by the `discord` command. In particular, it considers records having the breakend structural variant type (`SVTYPE=BND`) as defined in the VCF specification. Other types of record are ignored, but the `svdecompose` command can be applied beforehand to split certain other structural variants (e.g., `INV` and `DEL`) or sequence-resolved insertions and deletions into constituent breakend events.

The input and output requirements of `bndeval` are broadly similar to the `vcfeval` command. The primary inputs to `bndeval` are a truth/baseline VCF containing expected breakends, and a query/call VCF containing the called breakends. Evaluation can be restricted to particular regions by specifying a BED file.

The regions contained in the evaluation regions BED file are intersected with the breakend records contained in the truth VCF in order to obtain a list of *truth breakend regions*. An evaluation region is included if there is any overlapping truth VCF record (no attempt is made to look at the degree of overlap). Thus by supplying either evaluation regions corresponding to targeted regions or larger gene-level regions `bndeval` can be used to evaluate at different levels of granularity.

Similarly, the evaluation regions are intersected with the breakend records records contained in the calls VCF to obtain *called breakend regions*.

The *truth breakend regions* are then intersected with the *called breakend regions* to obtain TP/FP/FN metrics. The intersection supports a user-selectable tolerance in position. Further, be default, a breakend must occur in the same orientation to be considered a match, but this constraint can be relaxed by supplying the `--bidirectional` command line option.

bndeval outputs

Once complete, `bndeval` command produces summary statistics and the following primary result files in the output directory:

- `weighted_roc.tsv.gz` - contains ROC data that can be plotted with `rocplot`
- `baseline.bed.gz` contains the *truth breakend regions*, where each BED record contains the region status as `TP` or `FN`, the `SVTYPE`, and the span of the original truth VCF record.
- `calls.bed.gz` contains the *called breakend regions*, where each BED record contains the region status as `TP` or `FP`, the `SVTYPE`, the span of the original calls VCF record, and the score value used for ranking in the ROC plot.
- `summary.txt` contains the same summary statistics printed to standard output.

See also:

discord, *svdecompose*, *vcfeval*, *rocplot*

2.11.25 pedfilter

Synopsis:

Filter and convert a pedigree file.

Syntax:

```
$ rtg pedfilter [OPTION]... FILE
```

Example:

```
$ rtg pedfilter --remove-parentage mypedigree.ped
```

Parameters:

File Input/Output		
	`FILE`	The pedigree file to process, may be PED or VCF, use '-' to read from stdin.

Filtering		
	`--keep-family=STRING`	Keep only individuals with the specified family ID. May be specified 0 or more times, or as a comma separated list.
	`--keep-ids=STRING`	Keep only individuals with the specified ID. May be specified 0 or more times, or as a comma separated list.
	`--keep-primary`	Keep only primary individuals (those with a PED individual line / VCF sample column)
	`--remove-parentage`	Remove all parent-child relationship information.

Reporting		
	`--vcf`	Output pedigree in in the form of a VCF header rather than PED.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

The `pedfilter` command can be used to perform manipulations on pedigree information and convert pedigree information between PED and VCF header format. For more information about the PED file format see *Pedigree PED input file format*.

The VCF files output by the `family` and `population` commands contain full pedigree information represented as VCF header lines, and the `pedfilter` command allows this information to be extracted in PED format.

This command produces the pedigree output on standard output, which can be redirected to a file or another pipeline command as required.

See also:

family, *population*, *mendelian*, *pedstats*

2.11.26 pedstats

Synopsis:

Output information from pedigree files of various formats.

Syntax:

```
$ rtg pedstats [OPTION]... FILE
```

Example:

For a summary of pedigree information:

```
$ rtg pedstats ceph_pedigree.ped

Pedigree file: /data/ceph/ceph_pedigree.ped

Total samples:               17
Primary samples:             17
Male samples:                 9
Female samples:               8
Afflicted samples:            0
Founder samples:              4
Parent-child relationships:  26
Other relationships:          0
Families:                     3
```

To output a list of all founders:

```
$ rtg pedstats --founder-ids ceph_pedigree.ped
NA12889
NA12890
NA12891
NA12892
```

For quick pedigree visualization using GraphViz and ImageMagick, use a command-line such as:

```
$ dot -Tpng <(rtg pedstats --dot "A Title" mypedigree.ped) | display -
```

Parameters:

File Input/Output		
	FILE	The pedigree file to process, may be PED or VCF, use '-' to read from stdin.

Reporting		
-d	--delimiter=STRING	Output id lists using this separator (Default is \n)
	--dot=STRING	Output pedigree in GraphViz format, using the supplied text as a title.
	--families	Output information about family structures.
	--female-ids	Output ids of all females.
	--founder-ids	Output ids of all founders.
	--male-ids	Output ids of all males.
	--maternal-ids	Output ids of maternal individuals.
	--paternal-ids	Output ids of paternal individuals.
	--primary-ids	Output ids of all primary individuals.
	--simple-dot	When outputting GraphViz format, use a layout that looks less like a traditional pedigree diagram but works better with large complex pedigrees.

Utility		
-h	--help	Print help on command-line flag usage.

Usage:

This command is used to show pedigree summary statistics or select groups of individual IDs.

When using `pedstats` to output a list of sample IDs, the default is to print one ID per line. Depending on subsequent use, it may be convenient to use a different separator between output IDs. For example, with comma separated output it is possible to directly use the results as an argument to `vcfsubset`:

```
$ rtg vcfsubset -i pedigree-calls.vcf.gz -o family1.vcf.gz \
    --keep-samples <(rtg pedstats -d , --founder-ids ceph_pedigree.ped)
```

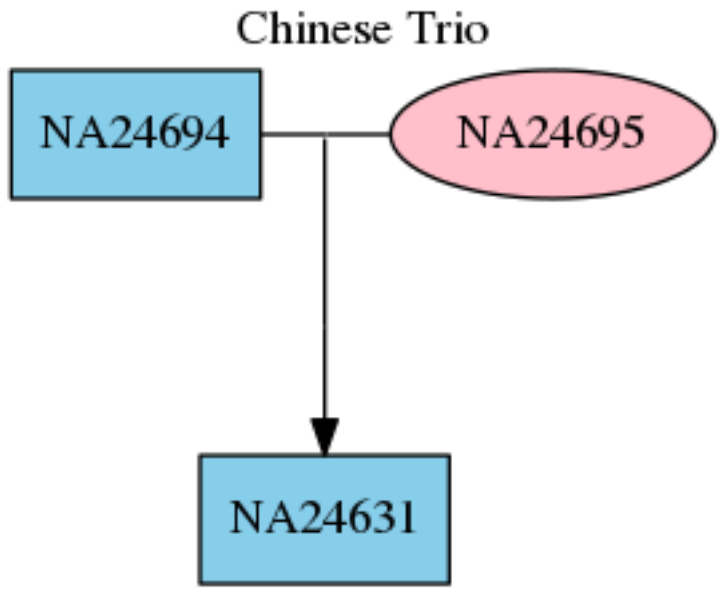
In addition, `pedstats` can be used to generate a simple pedigree visualization, using the well-known GraphViz graphics drawing package, which can be saved to PNG or PDF. For example, with the following `chinese-trio.ped`:

```
#PED format pedigree
#
#fam-id/ind-id/pat-id/mat-id: 0=unknown
#sex: 1=male; 2=female; 0=unknown
#phenotype: -9=missing, 0=missing; 1=unaffected; 2=affected
#
#fam-id ind-id  pat-id  mat-id  sex     phen
0       NA24631 NA24694 NA24695 1       0
0       NA24694 0       0       1       0
0       NA24695 0       0       2       0
```

We can visualize the pedigree with:

```
$ dot -Tpng <(rtg pedstats --dot "Chinese Trio" chinese-trio.ped) -o chinese-trio.
↪png
```

This will create a PNG image that can be displayed in any image viewing tool and contains the pedigree structure as shown below.

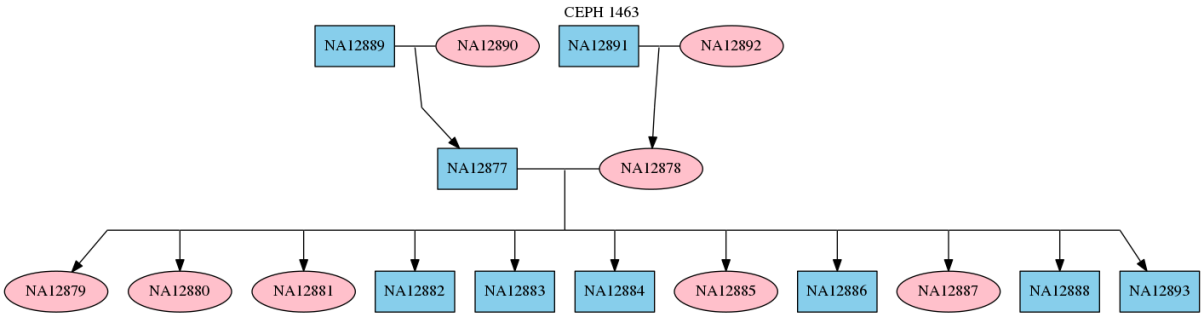


For more information about the PED file format see *Pedigree PED input file format*.

The VCF files output by the RTG pedigree-aware variant calling commands contain full pedigree information represented as VCF header lines, and the `pedstats` command can also take these VCFs as input. For example, given a VCF produced by the `population` command after calling the CEPH-1463 pedigree:

```
$ dot -Tpng <(rtg pedstats --dot "CEPH 1463" population-ceph-calls.vcf.gz) -o ceph-
↪1463.png
```

Would produce the following pedigree directly from the VCF:



Note: GraphViz is provided directly via many operating system package managers, and can also be downloaded from their web site: https://www.graphviz.org/

See also:

family, *population*, *pedfilter*, *vcfsubset*

2.11.27 avrstats

Synopsis:

Print statistics that describe an AVR model.

Syntax:

```
$ rtg avrstats [OPTION]... FILE
```

Example:

```
$ rtg avrstats avr.model
```

Parameters:

Reporting		
	`MODEL`	Name of AVR model to use when scoring variants.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

Used to show some simple information about the AVR model, including when the model was built and which predictor attributes were employed during the model build.

See also:

avrbuild, *avrpredict*, *snp*, *family*, *population*

2.11.28 rocplot

Synopsis:

Plot ROC curves from `readsimeval` and `vcfeval` ROC data files, either to an image, or using an interactive GUI.

Syntax:

```
$ rtg rocplot [OPTION]... FILE+
```

```
$ rtg rocplot [OPTION]... --curve STRING
```

Example:

```
$ rtg rocplot eval/weighted_roc.tsv.gz
```

Parameters:

File Input/Output		
	`--curve=STRING`	ROC data file with title optionally specified (path[=title]). May be specified 0 or more times.
	`--png=FILE`	If set, output a PNG image to the given file.
	`--svg=FILE`	If set, output a SVG image to the given file.
	`--zoom=STRING`	Show a zoomed view with the given coordinates, supplied in the form <xmax>,<ymax> or <xmin>,<ymin>,<xmax>,<ymax>
	`FILE+`	ROC data file. May be specified 0 or more times.

Reporting		
	`--hide-sidepane`	If set, hide the side pane from the GUI on startup.
	`--interpolate`	If set, interpolate curves at regular intervals.
	`--line-width=INT`	Sets the plot line width (Default is 2)
`-P`	`--precision-sensitivity`	If set, plot precision vs sensitivity rather than ROC.
	`--scores`	If set, show scores on the plot.
`-t`	`--title=STRING`	Title for the plot.

Utility		
`-h`	`--help`	Print help on command-line flag usage.

Usage:

Used to produce ROC plots from the ROC files produced by `readsimeval`, `bndeval` and `vcfeval`. By default this opens the ROC plots in an interactive viewer. On a system with only console access the plot can be saved directly to an image file using the either the `--png` or `--svg` parameter.

ROC data files may be specified either as direct file arguments to the command, or via the `--curve` flag. The former method is useful when selecting files using shell wild card globbing, and the latter method allows specifying a custom title for each curve, so use whichever method is most convenient.

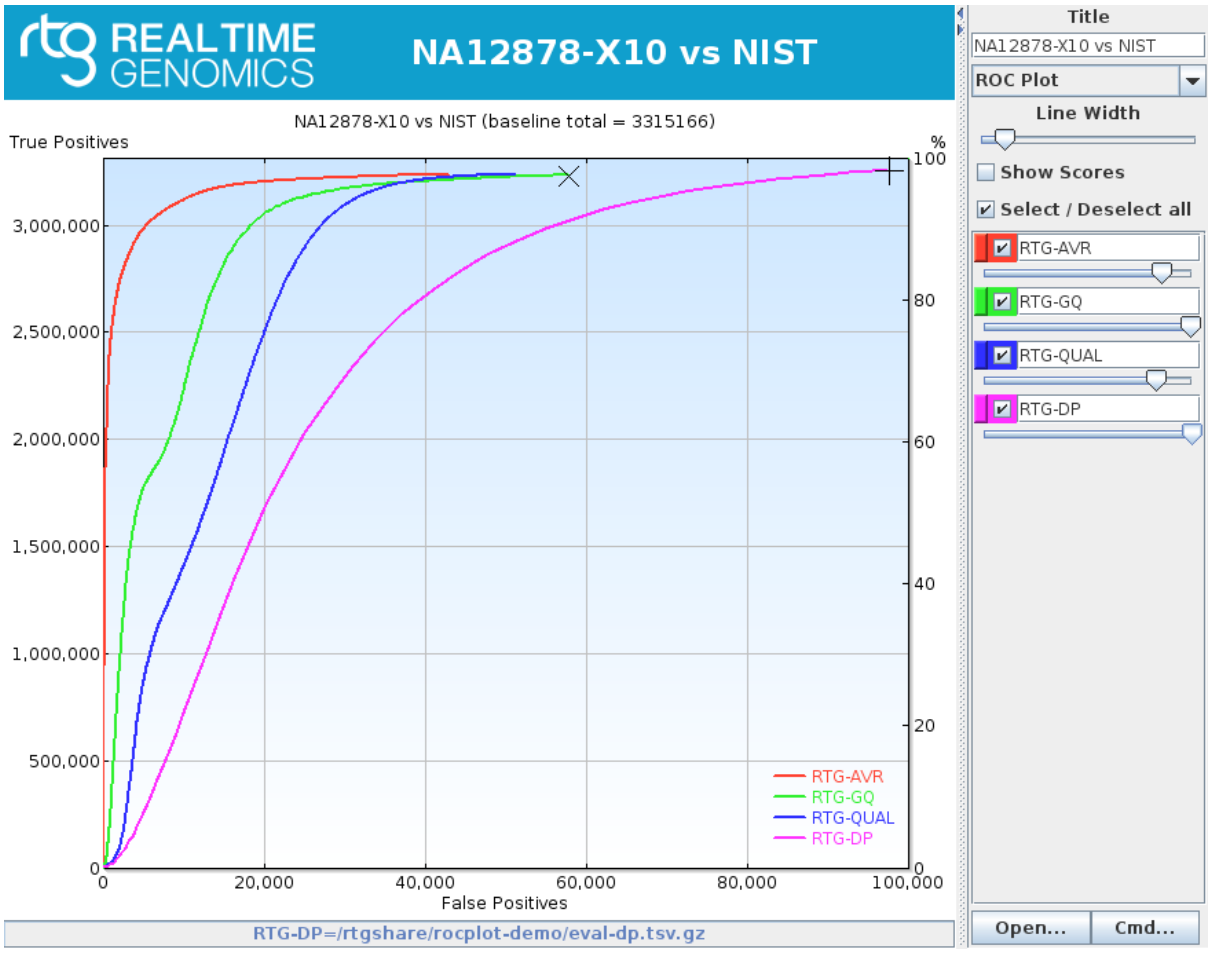
Strictly speaking, a true ROC curve should use *rates* rather than absolute numbers on the X and Y axes (e.g. True Positive / Total Positives rather than True Positives on the Y, and False Positive / Total Negatives on the X axis). However, there are a couple of difficulties involved with computing these rates with variant calling datasets. Firstly, the truth sets do not include any indication of the set of negatives (the closest we may get is in the cases of truth sets which contain a set of confidence regions, where it can be assumed that no other variants may be present inside the specified regions); secondly even with knowledge of negative regions, how do you count the set of possible negative calls, when a call could occupy multiple reference bases, or even (in the case of insertions) zero reference bases. It is conceptually even possible to have a call-set contain more false positives than there are reference bases. For this reason the ROC curves are plotted using the absolute counts.

Precision/sensitivity (also known as precision/recall) curves are another popular means of visualizing call-set accuracy, and these metrics also do not require a count of Total Negatives and so cause no particular difficulty to plot. Precision/sensitivity graphs can be selected from the command line via the `--precision-sensitivity` flag, or may be interactively selected in the GUI.

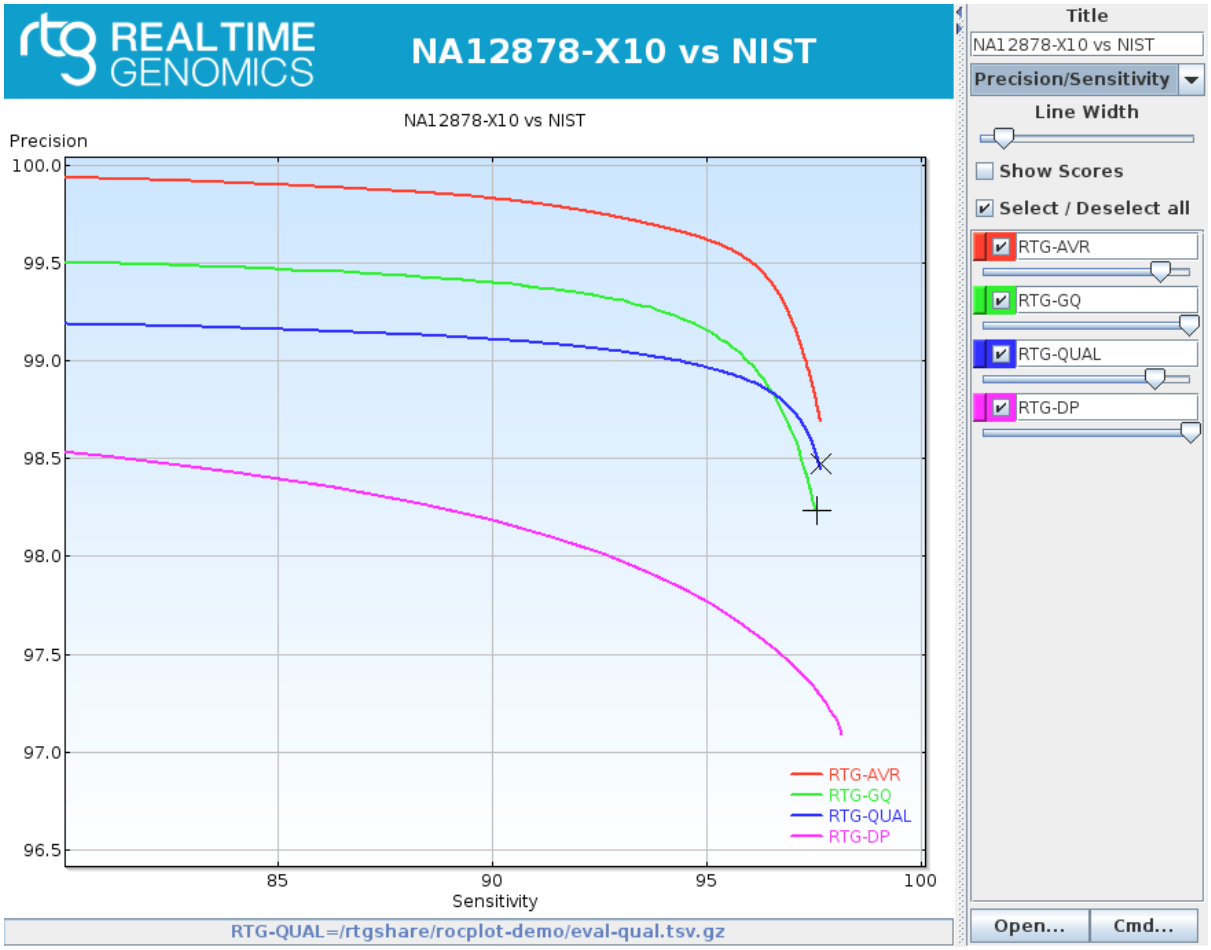
An interesting result of ROC analysis is that although there may be few data points on an ROC graph, it is possible to construct a filtered dataset corresponding to any point that lies on a straight line between two points on the graph. (For example, using threshold A for 25% of the variants and threshold B for 75% of the variants would result in accuracy that is 75% of the way between the points corresponding to thresholds A and B on the ROC plot). So in a sense it is meaningful to connect points on an ROC graph with straight lines. However, for precision/sensitivity graphs, it's incorrect to connect adjacent points with a straight line, as this does not correspond to achievable accuracy on the ROC convex hull and can over-estimate the accuracy. Instead, we can plot appropriately interpolated values with the `--interpolate` option.

Interactive GUI

The following image shows the `rocplot` GUI with an example ROC plot :



Similarly, here is an example precision/sensitivity plot:



Some quick tips for the interactive GUI:

- Select regions within the graph to zoom in. Right click within the graph area to bring up a context menu that allows undoing the zoom one level at a time, or resetting the zoom to the default.
- The graph right click menu also allows exporting the image as PNG or SVG. (The saved image does not include the RTG banner or background gradient).
- Click on a spot in the graph to show the equivalent accuracy metrics for that location in the status bar. Clicking to the left or below the axes will remove the cross-hair. Note that sensitivity depends on the baseline total number of variants being correct. If for example the ROC curve corresponds to evaluating an exome call-set against a whole-genome baseline, this number will be inaccurate.
- A secondary cross-hair is also available by holding down shift when placing (or removing) the cross-hair. When two cross-hairs are placed or moved, metrics in the status bar indicate the difference between the two positions.
- Additional ROC data files can be loaded by clicking on the "Open..." button, and multiple ROC data files within a directory can be loaded at once using multi-select.
- The "Cmd" button will open a message window that contains a command-line equivalent to the currently displayed set of curves. This command-line may be copy-pasted, providing an easy way to replicate the current set of curves in another session, generate a curve in a script, or share with a colleague.
- There is a drop down that allows for switching between ROC and precision/sensitivity graph types.

Each curve in the GUI has a customization widget on the right hand side of the window that allows several operations:

- Rename the title used for the curve via the editable text.
- Temporarily hide/show the curve via selection checkbox.

- Reorder curves via drag and drop using the colored handle on the left.
- Right click within the ROC widget area to bring up a context menu that allows permanently removing that curve, or customizing the color used for the curve
- Each curve has a slider to simulate the effect of applying a threshold on the scoring attribute. If the "show scores" option is set, this provides an easy way to select appropriate filter threshold values, which you might apply to variant sets using `rtg vcffilter` or similar VCF filtering tools.

Note: For definitions of the terminology used when evaluating caller accuracy, see: https://en.wikipedia.org/wiki/Receiver_operating_characteristic and https://en.wikipedia.org/wiki/Sensitivity_and_specificity

---

Note: For a description of the precision/sensitivity interpolation, see: "The relationship between Precision-Recall and ROC curves", Davis, J., (2006), https://dx.doi.org/10.1145/1143844.1143874

See also:

readsimeval, *bndeval*, *vcfeval*

2.11.29 hashdist

Synopsis:

Counts the number of times *k*-mers occur in an SDF and produces a histogram. Optionally creates a blacklist of highly occurring hashes that can be used to increase mapping speed

Syntax:

```
$ rtg hashdist [OPTION]... -o DIR SDF
```

Parameters:

File Input/Output		
`-o`	`--output=DIR`	Directory for output.
	`SDF`	SDF containing sequence data to analyse.

Sensitivity Tuning		
	`--blacklist-threshold=INT`	If set, output a blacklist containing all *k*-mer hashes with counts exceeding this value.
	`--hashmap-size-factor=FLOAT`	Multiplier for the minimum size of the hash map (Default is 1.0)
	`--max-count=INT`	Soft minimum for hash count (i.e. will record exact counts of at least this value) (Default is 500)
`-s`	`--step=INT`	Step size (Default is 1)
`-w`	`--word=INT`	Number of bases in each hash (Default is 22)

Utility		
`-h`	`--help`	Print help on command-line flag usage.
	`--install-blacklist`	Install the blacklist into the SDF for use during mapping.
`-T`	`--threads=INT`	Number of threads (Default is the number of available cores)

Usage:

Used to produce a text file containing a histogram of *k*-mer frequencies. The `--word` parameter is used to select the width of the *k*-mer and the `--step` parameter is used to select the distance between successive *k*-mer start positions.

By specifying the `--blacklist-threshold` parameter a list *k*-mers that occur more than the given number of times will be produced. Using the `--install-blacklist` option will install the resulting blacklist file into the specified SDF, which will permit use of the `--blacklist-threshold` parameter of the `map` command.

The `--max-count` parameter can be used to inexactly adjust memory requirements by setting a lower bound on the largest *k*-mer count that will be recorded. For example, `--max-count 500` will select a number greater than or equal to 500 (exactly how much greater will depend on other memory requirements), and will record exact frequencies for all *k*-mers than occur less than this number. All *k*-mers that occur more frequently than the chosen limit will be capped at the limit.

The `--hashmap-size-factor` parameter controls the default size of the internal hash map, which in turn affects the RAM required to run the command. This value may need to be increased if `hashdist` reports warnings about too many hash collisions. Alternatively this parameter could be reduced in order to run on a machine with lower RAM, but this may reduce the likelihood that the command will complete successfully.

See also:

map

2.11.30 ncbi2tax

Synopsis:

Converts the NCBI taxonomy into an RTG taxonomy for use in species database construction.

Syntax:

```
$ rtg ncbi2tax [OPTION]... DIR
```

Example:

```
$ wget ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
$ tar zxf taxdump.tar.gz -C ncbitaxdir
$ rtg ncbi2tax ncbitaxdir >rtg_taxonomy.tsv
```

Parameters:

File Input/Output		
	`DIR`	Directory containing the NCBI taxonomy dump.

Utility		
`-h`	`--help`	Prints help on command-line flag usage.

Usage:

Used to create an RTG taxonomy file from an NCBI taxonomy dump. The resulting taxonomy TSV file can be directly filtered with the `taxfilter` command prior to creating a species reference SDF according to project needs.

For more information on the RTG taxonomy format, and the associated sequence to taxon mapping file needed to create a species reference SDF, see *RTG taxonomic reference file format*.

See also:

format, *species*, *taxfilter*, *taxstats*

2.11.31 taxfilter

Synopsis:

Provides filtering of a metagenomic species reference database taxonomy.

Syntax:

```
$ rtg taxfilter [OPTION]... -i FILE -o FILE
```

Example:

```
$ rtg taxfilter -P -i species-full.sdf -o species-pruned.sdf
```

Parameters:

File Input/Output		
-i	--input=FILE	Taxonomy input. May be either a taxonomy TSV file or an SDF containing taxonomy information.
-o	--output=FILE	Filename for output TSV or SDF.

Filtering		
-P	--prune-below-internal-sequences	When filtering an SDF, remove nodes below the first containing sequence data.
-p	--prune-internal-sequences	When filtering an SDF, exclude sequence data from non-leaf output nodes.
-r	--remove=FILE	File containing ids of nodes to remove.
-R	--remove-sequences=FILE	File containing ids of nodes to remove sequence data from (if any).
	--rename-norank=FILE	Assign a rank to "no rank" nodes from file containing id/rank pairs.
-s	--subset=FILE	File containing ids of nodes to include in subset.
-S	--subtree=FILE	File containing ids of nodes to include as subtrees in subset.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

The `taxfilter` command is used to manage metagenomic species reference taxonomies and associated reference species SDF, primarily to allow redundancy reduction and extraction of a subset of the database according to project needs.

Building a metagenomic species database from all available data typically results in a very large database with high levels of redundancy, as often multiple strains of species are present, and often entire branches of the taxonomic structure are irrelevant for the project at hand. The following options provides methods to prune the taxonomy to sections of interest.

The `--remove` option will remove the specified taxon IDs from the taxonomy (along with any child nodes), which can be used to exclude entire subtrees of the full taxonomy. For example, you might exclude all Proteobacteria by specifying that `taxfilter` should remove node with taxon ID 1224.

The `--subset` option allows retaining only the specified list of taxon IDs, along with any parent nodes required to reach the root of the taxonomy. This would typically be used to specify a list of species or strains of interest.

The `--subtree` option allows retaining the specified nodes, along with their children and any parent nodes required to reach the root of the taxonomy. For example, you could retain all Firmicutes by specifying that `taxfilter` should keep the subtree with taxon ID 1239.

It is also often the case that ranks have not been fully assigned for each node in the taxonomic structure. The `--rename-norank` option allows manual rank assignment for any of these nodes for which rank information can be obtained via other means, such as manual curation.

The `species` command requires that the reference database not contain sequence data assigned to internal nodes of the taxonomy, so the application of `--prune-internal-sequences`, `--prune-below-internal-sequences`, or `--remove-sequences` may be required before using any such database with the `species` command. The `taxstats` command can be used to list the ids of internal taxons that have sequence data attached.

Note that a quick way to extract all the genomic sequence associated with a species (or multiple species) is to use the `sdf2fasta` command with the `--taxon` flag.

See also:

format, *sdf2fasta*, *species*, *taxstats*

2.11.32 taxstats

Synopsis:

Summarize and perform a verification of taxonomy and sequence information within a metagenomic species reference SDF.

Syntax:

```
$ rtg taxstats [OPTION]... SDF
```

Example:

```
$ rtg taxstats species-full.sdf
Warning: 340 nodes have no rank
214 nodes with no rank are internal nodes
126 nodes with no rank are leaf nodes
126 nodes with no rank have sequences attached
TREE STATS
internal nodes: 3724
leaf nodes:     5183
total nodes:    8907

RANK COUNTS
rank              internal leaf total
class                   58    0    58
family                 300    0   300
genus                  940    1   941
no rank                214  126   340
order                  127    0   127
phylum                  34    0    34
species               1709 1703  3412
species group           34    0    34
species subgroup         7    0     7
strain                 146 3347  3493
subclass                 5    0     5
subfamily               17    0    17
subgenus                 1    0     1
suborder                19    0    19
subphylum                1    0     1
subspecies             104    6   110
superkingdom             3    0     3
superphylum              3    0     3
tribe                    2    0     2
TOTAL                 3724 5183  8907

SEQUENCE LOOKUP STATS
total sequences:           309367
unique taxon ids:            5183
taxon ids in taxonomy:       5183
taxon ids not in taxonomy:      0
internal nodes:                 0
leaf nodes:                  5183
no rank nodes:                126
```

Parameters:

File Input/Output		
	SDF	SDF to verify the taxonomy information for.

Reporting		
	--show-details	List details of sequences attached to internal nodes of the taxonomy.

Utility		
-h	--help	Prints help on command-line flag usage.

Usage:

The `taxstats` command outputs statistics regarding the contents of a metagenomic species reference database, in order to indicate the number of members of each rank, and how many have sequence information contained within the database.

Any discrepancies found within the database will be issued as warnings.

See also:

format, *species*, *taxfilter*

2.11.33 usageserver

Synopsis:

Start a local network usage logging server.

Syntax:

```
$ rtg usageserver [OPTION]...
```

Example:

```
$ rtg usageserver
```

Parameters:

Utility		
-h	--help	Prints help on command-line flag usage.
-p	--port=INT	Set this flag to change which port to listen for usage logging connections. (Default is 8080).
-T	--threads=INT	Set this flag to change the number of threads for handling incoming connections. (Default is 4).

Usage:

Use the `usageserver` command to run a usage logging server for a local network. For more information about usage logging and setup see *Usage logging*

2.11.34 version

Synopsis:

The RTG version display utility.

Syntax:

```
$ rtg version
```

Example:

```
$ rtg version

Product: RTG Core 3.9
Core Version: 718f8317b7 (2018-05-29)
RAM: 25.0GB of 31.3GB RAM can be used by rtg (79%)
CPU: Defaulting to 4 of 4 available processors (100%)
JVM: Java HotSpot(TM) 64-Bit Server VM 1.8.0_161
License: Expires on 2019-05-20
Contact: support@realtimegenomics.com
```

```
Patents / Patents pending:
US: 7,640,256, 9,165,253, 13/129,329, 13/681,046, 13/681,215, 13/848,653, 13/925,
↪704, 14/015,295, 13/971,654, 13/971,630, 14/564,810
UK: 1222923.3, 1222921.7, 1304502.6, 1311209.9, 1314888.7, 1314908.3
New Zealand: 626777, 626783, 615491, 614897, 614560
Australia: 2005255348, Singapore: 128254

Citation (variant calling):
John G. Cleary, Ross Braithwaite, Kurt Gaastra, Brian S. Hilbush, Stuart Inglis,␣
↪Sean A. Irvine, Alan Jackson, Richard Littin, Sahar Nohzadeh-Malakshah, Mehul␣
↪Rathod, David Ware, Len Trigg, and Francisco M. De La Vega. "Joint Variant and␣
↪De Novo Mutation Identification on Pedigrees from High-Throughput Sequencing␣
↪Data." Journal of Computational Biology. June 2014, 21(6): 405-419. doi:10.1089/
↪cmb.2014.0029.

Citation (vcfeval):
John G. Cleary, Ross Braithwaite, Kurt Gaastra, Brian S. Hilbush, Stuart Inglis,␣
↪Sean A. Irvine, Alan Jackson, Richard Littin, Mehul Rathod, David Ware, Justin M.
↪ Zook, Len Trigg, and Francisco M. De La Vega. "Comparing Variant Call Files for␣
↪Performance Benchmarking of Next-Generation Sequencing Variant Calling Pipelines.
↪" bioRxiv, 2015. doi:10.1101/023754.

(c) Real Time Genomics, 2017
```

Parameters:

There are no options associated with the `version` command.

Usage:

Use the `version` command to display release and version information.

See also:

help, *license*

2.11.35 license

Synopsis:

The RTG license display utility.

Syntax:

```
$ rtg license
```

Example:

```
$ rtg license
```

Parameters:

There are no options associated with the license command.

Usage:

Use the `license` command to display license information and expiration date. Output at the command line (standard output) shows command name, licensed status, and command release level.

See also:

help, *version*

2.11.36 help

Synopsis:

The RTG help command provides online help for all RTG commands.

Syntax:

List all commands:

```
$ rtg help
```

Show usage syntax and flags for one command:

```
$ rtg help COMMAND
```

Example:

```
$ rtg help format
```

Parameters:

There are no options associated with the `help` command.

Usage:

Use the `help` command to view syntax and usage information for the main `rtg` command as well as individual RTG commands.

See also:

license, *version*

CHAPTER

THREE

RTG PRODUCT USAGE - BASELINE PROGRESSIONS

This chapter provides baseline task progressions that describe best-practice use of the product for data analysis.

3.1 Human read mapping and sequence variant detection

Use the following steps to detect all sequence variants between a reference genome and a sequenced DNA sample or set of related samples. This set of tasks steps through the main functionality of the RTG variant detection software pipeline: generating and evaluating gapped alignments, testing coverage depth, and calling sequence variants (SNPs, MNPs, and indels). While this progression is aimed at human variant detection, the process can easily accommodate other mammalian genomes. For non-mammalian species, some features such as sex and pedigree-aware mapping and variant calling may need to be omitted.

The following example supports the steps typical to human whole exome or whole genome analysis in which high-throughput sequencing with Illumina sequencing systems has generated reads of length 100 to 300 base pairs and at 25 to 30× genome coverage.

RTG features the ability to adjust mapping and variant calling according to the sex of the individual that has been sequenced. During mapping, reads will only be mapped against chromosomes that are present in that individuals genome (for example, for a female individual, reads will not be mapped to the Y chromosome). Similarly, sex-aware variant calling will automatically determine when to switch between diploid and haploid models. In addition, when pedigree information is available, Mendelian inheritance patterns are used to inform the variant calling.

This example demonstrates exome variant calling, automatic sex-aware calling capabilities, and joint variant calling with respect to a pedigree.

Data:

This baseline uses actual data downloaded from public databases. The reference sequence is the human GRCh38, downloaded from ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines.ucsc_ids/GCA_000001405.15_GRCh38_no_alt_plus_hs38d1_analysis_set.fna.gz This version excludes alternate haplotypes but includes the hs38d1 decoy sequence which improves variant calling accuracy by providing targets for reads with high homology to sites on the primary chromosomes.

The read data is from human genome sequencing of the CEPH pedigree 1463, which comprises 17 members across three generations. Illumina sequencing data is provided as part of the Illumina Platinum Genomes project, available via https://www.illumina.com/platinumgenomes.html and also available in the European Nucleotide Archive. Complete Genomics sequencing data for these samples is available via http://www.completegenomics.com/public-data/69-Genomes/

Table : Overview of basic pipeline stages

Task		Command & Utilities	Purpose
1	Format reference data	`rtg format`	Convert reference sequence from FASTA files to RTG Sequence Data Format (SDF)
2	Prepare pedigree/sex information	`rtg pedstats`	Configure per-sample sex and pedigree relationship information in a PED file
3	Format read data	`rtg format`, `rtg cg2sdf`	Convert read sequence from FASTA or FASTQ files to RTG Sequence Data Format (SDF)
4	Map reads against a reference genome	`rtg map`, `rtg cgmap`	Generate read alignments against a given reference, and report in a BAM file for downstream analysis
5	View alignment results	`rtg samstats`	Evaluate alignments and determine if the mapping should be repeated with different settings
6	Generate coverage information	`rtg coverage`	Run the coverage command to generate coverage breadth and depth statistics
7	Call sequence variants (single sample)	`rtg snp`	Detect SNPs, MNPs, and indels in a sample relative to a reference genome
8	Call sequence variants (single family)	`rtg family`	Perform sex-aware joint variant calls relative to the reference on a Mendelian family
9	Call sequence variants (population)	`rtg population`	Perform sex-aware joint variant calls relative to the reference on a population

3.1.1 Task 1 - Format reference data

RTG requires a one-time conversion of the reference genome from FASTA files into the RTG SDF format. This provides the software with fast access to arbitrary genomic coordinates in a pre-parsed format. In addition, metadata detailed autosomes and sex chromosomes is associated with the SDF, enabling automatic handing of sex chromosomes in later tasks. This task will be completed with the `format` command. The conversion will create an SDF directory containing the reference genome.

For variant calling and CNV analysis, the reference genome should ideally employ fully assembled chromosomes (as opposed to contig-based coordinates where expected ploidy and inheritance characteristics may be unknown).

First, observe a typical genome reference with multiple chromosomes stored in compressed FASTA format.

```
$ ls -l /data/human/grch38/fasta
874637925 Feb 28 11:00 GCA_000001405.15_GRCh38_no_alt_plus_hs38d1_analysis_set.fna.
↪gz
```

Now, use the `format` command to convert this input file into a single SDF directory for the reference genome. If the reference is comprised of multiple input files, these should be supplied on a single command line.

```
$ rtg format -o grch38 /data/human/grch38/fasta/GCA_000001405.15_GRCh38_no_alt_
↪plus_hs38d1_analysis_set.fna.gz
Formatting FASTA data
Processing "/data/human/grch38/fasta/GCA_000001405.15_GRCh38_no_alt_plus_hs38d1_
↪analysis_set.fna.gz"
Unexpected symbol "M" in sequence "chr1  AC:CM000663.2  gi:568336023  LN:248956422␣
↪ rl:Chromosome  M5:6aef897c3d6ff0c78aff06ac189178dd  AS:GRCh38" replaced with "N
↪".
[...]

Detected: 'Human GRCh38 with UCSC naming', installing reference.txt

Input Data
Files               : GCA_000001405.15_GRCh38_no_alt_plus_hs38d1_analysis_set.fna.gz
Format              : FASTA
Type                : DNA
Number of sequences: 2580
```

```
Total residues      : 3105715063
Minimum length      : 970
Maximum length      : 248956422

Output Data
SDF-ID              : c477b940-e9e7-4068-b7ed-f8968de308e3
Number of sequences: 2580
Total residues      : 3105715063
Minimum length      : 970
Maximum length      : 248956422
```

This takes the human reference FASTA file and creates a directory called `grch38` containing the SDF. If the reference genome contains IUPAC ambiguity codes or other non-DNA characters these are replaced with 'N' (unknown) in the formatted SDF. The first few such codes will generate warning messages as above. Every SDF contains a unique SDF-ID identifier that is reported in log files of subsequent commands and to help identify potential pipeline issues that arise from incorrectly using different human reference builds at different pipeline stages.

Note: When formatting a reference genome, the `format` command will automatically recognize several common human reference genomes and install a `reference.txt` configuration file. For reference genomes which are not recognized, you should copy or create an appropriate `reference.txt` file into the SDF directory if you plan on performing sex-aware mapping and variant calling. See *RTG reference file format*

You can use the `sdfstats` command to show statistics for your reference SDF, including the individual sequence lengths.

```
$ rtg sdfstats --lengths grch38
Type                : DNA
Number of sequences: 2580
Maximum length      : 248956422
Minimum length      : 970
Sequence names      : yes
Sex metadata        : yes
Taxonomy metadata   : no
N                   : 165046090
A                   : 868075685
C                   : 599957776
G                   : 602090069
T                   : 870545443
Total residues      : 3105715063
Residue qualities   : no

Sequence lengths:
chr1    248956422
chr2    242193529
chr3    198295559
chr4    190214555
chr5    181538259
chr6    170805979
chr7    159345973
chr8    145138636
chr9    138394717
chr10   133797422
chr11   135086622
chr12   133275309
chr13   114364328
chr14   107043718
chr15   101991189
chr16   90338345
chr17   83257441
```

```
chr18   80373285
chr19   58617616
chr20   64444167
chr21   46709983
chr22   50818468
chrX    156040895
chrY    57227415
chrM    16569
chr1_KI270706v1_random  175055
chr1_KI270707v1_random  32032
chr1_KI270708v1_random  127682
[...]
```

Note the presence of reference chromosome metadata on the line `Sex metadata`. This indicates that the information for sex-aware processing is present. you can use the `sdfstats` command to verify the chromosome ploidy information for each sex. For the male sex chromosomes the PAR region mappings are also displayed.

```
$ rtg sdfstats grch38 --sex male --sex female
[...]
Sequences for sex=MALE:
chr1 DIPLOID linear 248956422
chr2 DIPLOID linear 242193529
chr3 DIPLOID linear 198295559
chr4 DIPLOID linear 190214555
chr5 DIPLOID linear 181538259
chr6 DIPLOID linear 170805979
chr7 DIPLOID linear 159345973
chr8 DIPLOID linear 145138636
chr9 DIPLOID linear 138394717
chr10 DIPLOID linear 133797422
chr11 DIPLOID linear 135086622
chr12 DIPLOID linear 133275309
chr13 DIPLOID linear 114364328
chr14 DIPLOID linear 107043718
chr15 DIPLOID linear 101991189
chr16 DIPLOID linear 90338345
chr17 DIPLOID linear 83257441
chr18 DIPLOID linear 80373285
chr19 DIPLOID linear 58617616
chr20 DIPLOID linear 64444167
chr21 DIPLOID linear 46709983
chr22 DIPLOID linear 50818468
chrX HAPLOID linear 156040895 ~=chrY
    chrX:10001-2781479  chrY:10001-2781479
    chrX:155701383-156030895  chrY:56887903-57217415
chrY HAPLOID linear 57227415 ~=chrX
    chrX:10001-2781479  chrY:10001-2781479
    chrX:155701383-156030895  chrY:56887903-57217415
chrM POLYPLOID circular 16569
chr1_KI270706v1_random DIPLOID linear 175055
[...]

Sequences for sex=FEMALE:
chr1 DIPLOID linear 248956422
chr2 DIPLOID linear 242193529
chr3 DIPLOID linear 198295559
chr4 DIPLOID linear 190214555
chr5 DIPLOID linear 181538259
chr6 DIPLOID linear 170805979
chr7 DIPLOID linear 159345973
chr8 DIPLOID linear 145138636
chr9 DIPLOID linear 138394717
```

```
chr10 DIPLOID linear 133797422
chr11 DIPLOID linear 135086622
chr12 DIPLOID linear 133275309
chr13 DIPLOID linear 114364328
chr14 DIPLOID linear 107043718
chr15 DIPLOID linear 101991189
chr16 DIPLOID linear 90338345
chr17 DIPLOID linear 83257441
chr18 DIPLOID linear 80373285
chr19 DIPLOID linear 58617616
chr20 DIPLOID linear 64444167
chr21 DIPLOID linear 46709983
chr22 DIPLOID linear 50818468
chrX DIPLOID linear 156040895
chrY NONE linear 57227415
chrM POLYPLOID circular 16569
chr1_KI270706v1_random DIPLOID linear 175055
[...]
```

3.1.2 Task 2 - Prepare sex/pedigree information

RTG commands for mapping and variant calling multiple samples make use of a pedigree file specifying sample names, their sex, and any relations between them. This is done by creating a standard PED format file containing information about the individual samples using a text editor. For example a PED file for the CEPH 1463 pedigree looks like the following.

```
$ cat population.ped
# PED format pedigree for CEPH/Utah 1463
# fam-id        ind-id  pat-id  mat-id  sex     phen
1       NA12889 0       0       1       0
1       NA12890 0       0       2       0
1       NA12877 NA12889 NA12890 1       0
2       NA12891 0       0       1       0
2       NA12892 0       0       2       0
2       NA12878 NA12891 NA12892 2       0
3       NA12879 NA12877 NA12878 2       0
3       NA12880 NA12877 NA12878 2       0
3       NA12881 NA12877 NA12878 2       0
3       NA12882 NA12877 NA12878 1       0
3       NA12883 NA12877 NA12878 1       0
3       NA12884 NA12877 NA12878 1       0
3       NA12885 NA12877 NA12878 2       0
3       NA12886 NA12877 NA12878 1       0
3       NA12887 NA12877 NA12878 2       0
3       NA12888 NA12877 NA12878 1       0
3       NA12893 NA12877 NA12878 1       0
```

A value of 0 indicates the field is unknown or that the sample is not present. Note that the IDs used in columns 2, 3, and 4 must match the sample IDs used during data formatting and variant calling. The sex field (column 5) may be 0 if the sex of a sample is unknown. The family ID in column 1 is ignored as RTG identifies family units according to the relationship information. You can use the `pedstats` command to verify the correct format:

```
$ rtg pedstats ceph-1463.ped
Pedigree file: ceph-1463.ped
Total samples:               17
Primary samples:             17
Male samples:                 9
Female samples:               8
Afflicted samples:            0
Founder samples:              4
```

```
Parent-child relationships:  26
Other relationships:          0
Families:                     3
```

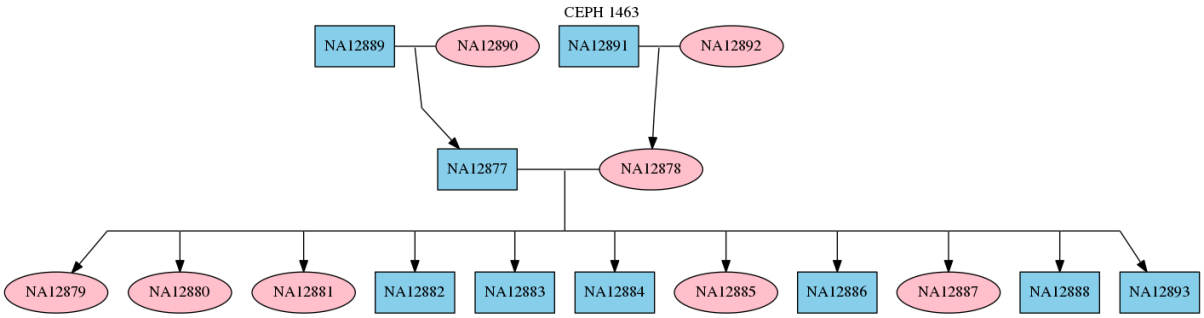
The `pedstats` command can also output the pedigree structure in a format that can be displayed using the `dot` command from the `Graphviz` suite.

```
$ dot -Tpng <(rtg pedstats --dot "CEPH 1463" ceph-1463.ped) -o ceph-1463.png
```

This will create a PNG image that can be displayed in any image viewing tool.

```
$ firefox ceph-1463.png
```

This will display the pedigree structure as shown below.



For more information about the PED file format see *Pedigree PED input file format*.

3.1.3 Task 3 - Format read data

RTG mapping tools accept input read sequence data either in the RTG SDF format or other common sequence formats such as FASTA, FASTQ, or SAM/BAM. There are pros and cons as to whether to perform an initial format of the read sequence data to RTG SDF:

- Pre-formatting requires an extra one-off workflow step (the `format` command), whereas native input file formats are directly accepted by many RTG commands.
- Pre-formatting requires extra disk space for the SDF (although these can be deleted after processing if required).
- With pre-formatting, decompression, parsing and error checking raw files is carried out only once, whereas native formats require this processing each time.
- Pre-formatting permits random access to individual sequences or blocks of sequences, whereas with native formats, the whole file leading up to the region of interest must also be decompressed, and parsed.
- Pre-formatting permits loading of sequence data, sequence names, and sequence quality values independently, allowing reduced RAM use during mapping

Thus, pre-formatting read sequence data can result in lower overall resource requirements (and faster throughput) than processing native file formats directly.

In this example we will be converting read sequence data from FASTQ files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create one or more SDF directories for the reads.

Take a paired set of reads in FASTQ format and convert it into RTG data format (SDF). This example shows one lane of data, taking as input both left and right paired files from the same run.

```
$ mkdir sample_NA12878
$ rtg format -f fastq -q sanger -o sample_NA12878/NA12878_L001 \
    -l /data/reads/NA12878/NA12878_L001_1.fq.gz \
```

```
    -r /data/reads/NA12878/NA12878_L001_2.fq.gz \
    --sam-rg "@RG\\tID:NA12878_L001\\tSM:NA12878\\tPL:ILLUMINA"
```

This creates a directory named `NA12878_L001` with two subdirectories, named `left` and `right`. Use the `sdfstats` command to verify this step.

```
$ rtg sdfstats sample_NA12878/NA12878_L001
```

It is good practice to ensure the output BAM files contain tracking information regarding the read set. This is achieved by storing the tracking information in the reads SDF by using the `--sam-rg` flag to provide a SAM-formatted read group header line. The header should specify at least the `ID`, `SM` and `PL` fields for the read group. The platform field (`PL`) values currently recognized by the RTG command for this attribute are `ILLUMINA`, `LS454`, `IONTORRENT`, `COMPLETE`, and `COMPLETEGENOMICS`. This information will automatically be used during mapping to enable automatic creation of calibration files that are used to perform base-quality recalibration during variant calling. In addition, the sample field (`SM`) is used to segregate samples when using multi-sample variant calling commands such as `family`, `population`, and `somatic`, and should correspond to the sample ids used in the corresponding pedigree file. The read group id field (`ID`) is a unique identifier used to group of reads that have the same source DNA and sequencing characteristics (for example, pertain to the same sequencing lane for the sample). For more details see *Using SAM/BAM Read Groups in RTG map*.

If the read group information is not specified when formatting the reads, it can be set during mapping instead using the `--sam-rg` flag of `map`.

Repeat this step for all available read data associated with the sample or samples to be processed. This example shows how this can be done with the `format` command in a loop.

```
$ for left_fq in /data/reads/NA12878/NA12878*_1.fq.gz; do
    right_fq=${left_fq/_1.fq.gz/_2.fq.gz}
    lane_id=$(basename ${left_fq/_1.fq.gz})
    rtg format -f fastq -q sanger -o ${lane_id} -l ${left_fq} -r ${right_fq} \
      --sam-rg "@RG\tID:${lane_id}\tSM:NA12878\tPL:ILLUMINA"
  done
```

RTG supports the custom read structure of Complete Genomics data. In this case, use the `cg2sdf` command to convert the GGI `.tsv` files into RTG SDF format. This example shows one run of data, taking as input a CGI read data file. It is important when mapping Complete Genomics reads that your read group information should set the platform field (`PL`) appropriately. For version 1 reads (35 base-pair reads), the platform should be `COMPLETE`. For version 2 reads (29 or 30 base-pair reads) the platform should be `COMPLETEGENOMICS`.

```
$ rtg cg2sdf -o sample_NA12878/NA12878_GS002290 /data/reads/cg/NA12878/GS002290.
↪tsv \
    --sam-rg "@RG\tID:GS002290\tSM:NA12878\tPL:COMPLETE"
```

As with the formatting of Illumina data, this creates a directory named `GS00290` with two subdirectories, named `left` and `right`. You can use the `sdfstats` command to verify this step.

```
$ rtg sdfstats sample_NA12878/NA12878_GS002290
```

Notice that during formatting we used the same sample identifier for both the Illumina and Complete Genomics data, which allows subsequent variant calling to associate both read sets with the NA12878 individual.

Repeat the formatting for other samples to be processed. When formatting data sets corresponding to other members of the pedigree, be sure to use the correct sample identifier for each individual.

3.1.4 Task 4 - Map reads to the reference genome

Map the sequence reads against the human reference genome to generate alignments in the BAM (Binary Sequence Alignment/Map) file format.

The RTG `map` command provides multiple tuning parameters to adjust sensitivity and selectivity at the read mapping stage. In general, whole genome mapping strategies aim to capture the highest number of reads possessing

variant data and which map with a high degree of specificity to the human genome. Complete Genomics data has particular mapping requirements, and for this data use the separate `cgmap` command.

Depending on the downstream analysis required, the mapping may be adjusted to restrict alignments to unique genomic positions or to allow reporting of ambiguous mappings at equivalent regions throughout the genome. This example will show the recommended steps for human genome analysis.

By default, the mapping will report positions for all mated pairs and unmated reads that map to the reference genome to 5 or fewer positions, those with no good alignments or with more than 5 equally good alignment positions are considered unmapped and are flagged as such in the output.

Note that when mapping each individual we supply the pedigree file with `--pedigree` to automatically determine the appropriate sex to be used when mapping each sample. For a female sample, this will result in no mappings being made to the Y chromosome. It is also possible to manually specify the sex of each sample via `--sex=male` or `--sex=female` as appropriate, but using the pedigree file streamlines the process across all samples.

For the whole-genome Illumina data, a suitable `map` command is:

```
$ mkdir map_sample_NA12878
$ rtg map -t grch38 --pedigree ceph-1463.ped \
    -i sample_NA12878/NA12878_L001 -o map_sample_NA12878/NA12878_L001 \
```

For Complete Genomics reads we use the `cgmap` command.

```
$ mkdir cgmap_sample_NA12878
$ rtg cgmap -t grch38 --pedigree ceph-1463.ped --mask cg1 \
    -i sample_NA12878/NA12878_GS002290 -o map_sample_NA12878/NA12878_GS002290 \
```

The selection of the `--mask` parameter depends on the length of the Complete Genomics reads to be mapped. The `cg1` and `cg1-fast` masks are appropriate for 35 base-pair reads and the `cg2` mask is appropriate for 29 or 30 base-pair reads. See *cgmap* for more detail.

When the mapping command completes, multiple files will have been written to the output directory of the mapping run. By default the `alignments.bam` file is produced and a BAM index (`alignments.bam.bai`) is automatically created to permit efficient extraction of mappings within particular genomic regions. This behavior is necessary for subsequent analysis of the mappings, but can be performed manually using the index command.

During mapping RTG automatically creates a calibration file (`alignments.bam.calibration`) containing information about base qualities, average coverage levels, etc. This calibration information is utilized during variant calling to improve accuracy and to determine when coverage levels are abnormally high. When processing exome or other targeted data, it is important that this calibration information should only be computed for mappings within the exome capture regions, otherwise the computed typical coverage levels will be much lower than actual. This can result in RTG discarding many variant calls as being over-coverage. The correct workflow for exome processing is to specify `--bed-regions` to supply a BED file describing the exome regions at the same time as mapping, to ensure appropriate calibration is computed.

```
$ rtg map -t grch38 --pedigree ceph-1463.ped --bed-regions exome-regions.bed \
    -i sample_NA12878/NA12878_L001 -o map_sample_NA12878/NA12878_L001 \
```

Note that supplying a BED file does restrict the locations to which reads are mapped, it is only used to ensure calibration information is correctly calculated.

Note: The exome capture BED file must correspond to the correct reference you are mapping and calling against. You may need to run the BED file supplied by your sequencing vendor through a lift-over tool if the reference genome versions differ.

The size of the job should be tuned to the available memory of the server node. You can perform mapping in segments by using the `--start-read` and `--end-read` flags during mapping. Currently, a 48 GB RAM system as specified in the technical requirements section can process 100 million reads in about an hour. The following example would work to map a data set containing just over 100 million reads in batches of 10 million.

```
$ for ((j=0;j<10;j++)); do
    rtg map -t grch38 --pedigree ceph-1463.ped \
      -i sample1-100M -o map_sample1-$j \
      --start-read=$((j*10000000)) --end-read=$(((j+1)*10000000))
  done
```

Note that each of these runs can be executed independently of the others. This allows parallel processing in a compute cluster that can reduce wall clock time.

In a parallel compute cluster special consideration is needed with respect to where data resides during the mapping process. Reading and writing data from and to a single networked disk partition may result in undesirable I/O performance characteristics depending on the size and structure of the compute cluster. One way to minimize the adverse affects of I/O limitations is to separate input data sets and map output directories, storing them on different network disk partitions.

Using the `--tempdir` flag allows the `map` command to use a directory other than the output directory to store temporary files that are output during the mapping process. The size of the temporary files is the same as the total size of files in the map output directory after processing has finished. The following example shows how to modify the above example to place outputs on a separate partition to the inputs.

```
$ mkdir /partition3/map_temp_sample1
$ for ((j=0;j<10;j++)); do
    rtg map -t /partition1/grch38 --pedigree /partition1/ceph-1463.ped \
      -i /partition1/sample1-100M \
      -o /partition2/map_sample1-$j \
      --start-read=$((j*1000000)) --end-read=$(((j+1)*1000000)) \
      --tempdir /partition3/map_temp_sample1
  done
```

Processing other samples in the pedigree is a matter of specifying different input SDF and output directories.

3.1.5 Task 5 - View and evaluate mapping performance

An `alignments.bam` file can be viewed directly using the RTG `extract` command in conjunction with a shell command such as `less` to quickly inspect the results.

```
$ rtg extract map_sample_NA12878/NA12878_L001-0/alignments.bam | less -S
```

Since the mappings are indexed by default, it is also possible to view mappings corresponding to particular genomic regions. For example:

```
$ rtg extract map_sample_NA12878/NA12878_L001-0/alignments.bam chr6:1000000-
↪5000000 \
  | less -S
```

The mapping output directory also contains a simple HTML summary report giving information about mapping counts, alignment score distribution, paired-end insert size distribution, etc. This can be viewed in your web browser. For example:

```
$ firefox map_sample_NA12878/NA12878_L001-0/index.html
```

For more detailed summary statistics, use the `samstats` command. This example will report information such as total records, number unmapped, specific details about the mate pair reads, and distributions for alignment scores, insert sizes and read hits.

```
$ rtg samstats -t grch38 map_sample_NA12878/NA12878_L001-0/alignments.bam \
   -r sample_NA12878/NA12878_L001 --distributions
```

Finally, the short summary of mapping produced on the terminal at the end of the `map` run, is also available as `summary.txt` in the mapping output directory.

3.1.6 Task 6 - Generate and review coverage information

For human genomic analysis, it is important to have sufficient coverage over the entire genome to detect variants accurately and minimize false negative calls. The `coverage` command provides detailed statistics for depth of coverage and gap size distribution. If coverage proves to be inadequate or spotty, remapping the data with different sensitivity tuning or rerunning the sample with different sequencing technology may help.

This example shows the `coverage` command used with all alignments, both mated and unmated, for the entire sample. The `-s` flag is used to introduce smoothing of the data, by default the data will not be smoothed. While you can supply the `coverage` command with the names of BAM files individually on the command line, this becomes unwieldy when the mapping has been carried out in many smaller jobs. In this command we will use the `-I` flag to supply a text file containing the names of all the mapping output BAM files, one per line. One example way to create this file is with the following command (assuming all your mapping runs have used a common root directory):

```
$ find map_sample_NA12878 -name "*.bam" > NA12878-bam-files.txt
$ rtg coverage -t grch38 -s 20 -o cov_sample_NA12878 -I NA12878-bam-files.txt
```

For exome or other targeted data, the coverage command can be instructed to focus only within the target regions by supplying the target regions BED file.

```
$ rtg coverage -t grch38 --bed-regions exome-regions.bed -s 20 \
    -o cov_sample_NA12878 -I NA12878-bam-files.txt
```

By default the `coverage` command will generate a BED formatted file containing regions of similar coverage. This BED file can be loaded into a genome browser to visualize the coverage, and may also be examined on the command line. For example:

```
$ rtg extract cov_sample_NA12878/coverage.bed.gz 'chr6:1000000-5000000' | less
```

The `coverage` command also creates a simple HTML summary report containing graphs of the depth of coverage distribution and cumulative depth of coverage. This can be viewed in your web browser. For example:

```
$ firefox cov_sample_NA12878/index.html
```

The `coverage` command has several options that alter the way that coverage levels are reported (for example, determining callability or giving per-region reporting). See *coverage* for more detail.

3.1.7 Task 7 - Call sequence variants (single sample)

The primary germline variant calling commands in the RTG suite are the `snp` command for detecting variants in a single sample, the `family` command for performing simultaneous joint calling of multiple family members within a single family, and the `population` command for performing simultaneous joint calling of multiple population members with varying degrees of relatedness. Where possible, we recommend using joint calling, as this ensures the maximum use of pedigree information as well as preventing cross-sample variant representation inconsistency that can occur when calling multiple samples individually. All of the RTG variant callers are sex-aware, producing calls of appropriate ploidy on the sex chromosomes and within PAR regions.

In this section, we demonstrate single sample calling with `snp`. For family calling, see *Task 8 - Call sequence variants (single family)*, and for population calling, see *Task 9 - Call population sequence variants*.

The `snp` command detects sequence variants in a single sample, given adequate but not excessively high coverage of reads against the reference genome. As with the `coverage` command, we can supply a file containing a list of all the needed input files. In this case the mapping calibration files will be automatically detected at the location of the BAM files themselves and will be used in order to enable base quality recalibration during variant calling.

This example takes all available BAM files for the sample and calls SNP, MNP, and indel sequence variants.

```
$ rtg snp -t grch38 --pedigree ceph-1463.ped \
    -o snp_sample_NA12878 -I sample_NA12878-bam-files
```

Here we supply the pedigree file in order to inform the variant caller of the sample sex. While multiple samples may be listed in the pedigree file, the `snp` command will object if mappings for multiple samples are supplied as input.

For exome data it is also recommended to provide the BED file describing the exome regions to the variant caller in order to filter the output produced to be within the exome regions. There are two approaches here. The first is to instruct the variant caller to only perform variant calling at sites within the exome regions by supplying the `--bed-regions` option:

```
$ rtg snp -t grch38 --pedigree ceph-1463.ped --bed-regions exome-regions.bed \
    -o snp_sample_NA12878 -I sample_NA12878-bam-files
```

This approach is computationally more efficient. However, if calls at off-target sites are potentially of interest, a second approach is to call variants at all sites but automatically filter variants as being off-target in the VCF `FILTER` field by using the `--filter-bed` option:

```
$ rtg snp -t grch38 --pedigree ceph-1463.ped --filter-bed exome-regions.bed \
    -o snp_sample_NA12878 -I sample_NA12878-bam-files
```

If you do not have a BED file that specifies the exome capture regions for your reference genome, you should supply the `--max-coverage` flag during variant calling to indicate an appropriate coverage threshold for over-coverage situations. A typical value is 5 times the expected coverage level within exome regions.

The `snp` command will perform the Bayesian variant calling and will use a default AVR model for scoring variant call quality. If you have a more appropriate model available, you should supply this with the `--avr-model` flag. RTG supplies some pre-built models which work well in a wide variety of cases, and includes tools for building custom AVR models. See *AVR scoring using HAPMAP for model building* for more information on building custom models.

The `snp` command will output variant call summary information upon completion, and this is also available in the output directory in the file `summary.txt`.

Variants are produced in standard VCF format. Since the variant calls are compressed and indexed by default, it is possible to view calls corresponding to particular genomic regions. For example:

```
$ rtg extract snp_sample_NA12878/snps.vcf.gz chr6:1000000-5000000 | less -S
```

Inspecting the output VCF for this run will show that no variants have been called for the Y chromosome. If you carry out similar sex-aware calling for the father of the trio, NA12891, inspection of the output VCF will show haploid variant calls for the X and Y chromosomes.

```
$ rtg extract snp_sample_NA12891/snps.vcf.gz chrX | head
$ rtg extract snp_sample_NA12891/snps.vcf.gz chrY | head
```

RTG also supports automatic handling of pseudoautosomal regions (PAR) and will produce haploid or diploid calls within the X PAR regions as appropriate for the sex of the individual.

A simple way to improve throughput is to call variants using a separate job for each reference sequence. Each of these runs could be executed on independent nodes of a cluster in order to reduce wall clock time.

```
$ for seq in M {1..22} X Y; do
    rtg snp -t grch38 --pedigree ceph-1463.ped -I sample_NA12878-bam-files \
      -o snp_sample_NA12878_chr"$seq" --region chr"$seq"
  done
```

After the separate variant calling jobs complete, the VCF files for each chromosome can be combined into a single file, if desired, by using a `vcfmerge` command such as:

```
$ rtg vcfmerge -o snp_sample_NA12878.vcf.gz snp_sample_NA12878_chr*/snps.vcf.gz
```

The individual `snp` commands will have output summary information about the variant calls made in each job. Combined summary information can be output for the merged VCF file with the `vcfstats` command.

```
$ rtg vcfstats snp_sample_NA12878.vcf.gz
```

Simple filtering of variants can be applied using the `vcffilter` command. For example, filtering calls by genotype quality or AVR score can be accomplished with a command such as:

```
$ rtg vcffilter --min-genotype-quality 50 -i snp_sample_NA12878.vcf.gz \
  -o filtered_sample_NA12878.vcf.gz
```

In this case, any variants failing the filter will be removed. Alternatively, failing variants can be kept but marked with a custom VCF `FILTER` field, such as:

```
$ rtg vcffilter --fail LOW-GQ --min-genotype-quality 50 \
  -i snp_sample_NA12878.vcf.gz -o filtered_sample_NA12878.vcf.gz
```

3.1.8 Task 8 - Call sequence variants (single family)

This section demonstrates joint calling multiple samples comprising a single Mendelian family. RTG is unique in supporting family calling beyond a single-child trio, and directly supports jointly calling a family involving multiple offspring. In this example, we call the large family of the CEPH pedigree containing 11 children.

The `family` command is invoked similarly to the `snp` command except you need to specify the way the samples relate to each other. This is done by specifying the corresponding sample ID used during mapping to the command line flags `--mother`, `--father`, `--daughter` and `--son`.

Note: The RTG `family` command only supports a basic family relationship of a mother, father and one or more children, either daughters or sons. For other pedigrees, use the `population` command.

To run the `family` command on the trio of NA12892 (mother), NA12891 (father) and NA12878 (daughter) you need to provide all the mapping files for the samples. The mapping calibration files will be automatically detected at the locations of the mapping files. To specify these in a file list for input you could run:

```
$ find /partition2/map_trio -name "alignments.bam" > map_trio-bam-files
```

To run the `family` command you then specify the sample ID for each member of the trio to the appropriate flag.

```
$ rtg family --mother NA12892 --father NA12891 --daughter NA12878 -t grch38 \
  -o trio_variants -I map_trio-bam-files
```

Examining the `snps.vcf.gz` file in the output directory will show individual columns for the variants of each family member. For more details about the VCF output file see *Small-variant VCF output file description*

Note: Per-family relationship information can also be specified using a pedigree PED file with the `--pedigree` flag. In this case, the pedigree file should contain a single family only.

3.1.9 Task 9 - Call population sequence variants

This section demonstrates joint variant calling of multiple potentially related or unrelated individuals in a population.

The `population` command is invoked similarly to the `snp` command except you must specify the pedigree file containing information about each sample and the relationships (if any) between them.

To run the `population` command on the population of NA12892, NA12891, NA19240 and NA12878 you need to provide all the mapping files for the samples. The mapping calibration files will be automatically detected at the locations of the mapping files. To specify these in a file list for input you could run:

```
$ find /partition2/map_population -name "alignments.bam" > map_population-bam-files
```

To run the `population` command you specify the PED file containing the sample ID for each member of the population in the individual ID column.

```
$ rtg population -t grch38 --pedigree ceph-1463.ped -o pop_variants \
    -I map_population-bam-files
```

Examining the `snps.vcf.gz` file in the output directory will show individual columns for the variants of each member of the population. For more details about the VCF output file see *Small-variant VCF output file description*.

3.2 Create and use population priors in variant calling

To improve the accuracy of variant calling on new members of a population, a file containing the allele counts of the population's known variants may be supplied. This information is used as an extra set of prior probabilities when making calls.

Sources of this allele count data can be external, for instance the 1000 genomes project, or from prior variant calling on other members of the population. An example use case of the latter follows.

Data

For this use case it is assumed that the following data is available:

- `/data/runs/20humans.vcf.gz` - output from a previous population command run on 20 humans from a population.
- `/data/reference/human_reference` - SDF containing the human reference sequences.
- `/data/mappings/new_human.txt` - text file containing a list of BAM files with the sequence alignments for the new member of the population.

Table : Overview of pipeline tasks.

Task		Command & Utilities	Purpose
1	Produce population priors file	`rtg vcfannotate`, `rtg vcfsubset`	Produce a reusable set of population priors from an existing VCF file
2	Run variant calling using population priors	`rtg snp`	Perform variant calling on the new member of the population using the new population priors to improve results

3.2.1 Task 1 - Produce population priors file

Using a full VCF file for a large population as population priors can be slow, as it contains lots of unnecessary information. The `AC` and `AN` fields are the standard VCF specification fields representing the allele count in genotypes, and total number of alleles in called genotypes. For more information on these fields, see the VCF specification at https://samtools.github.io/hts-specs/VCFv4.1.pdf. Alternatively, retaining only the `GT` field for each sample is sufficient, however this is less efficient both computationally and size-wise.

To calculate and annotate the `AC` and `AN` fields for a VCF file, use the RTG command `vcfannotate` with the parameter `--fill-an-ac`:

```
$ rtg vcfannotate --fill-an-ac -i /data/runs/20humans.vcf.gz \
  -o 20humans_an_ac.vcf.gz
```

Then remove all unnecessary data from the file using the RTG command `vcfsubset` as follows:

```
$ rtg vcfsubset --keep-info AC,AN --remove-samples \
  -i 20humans_an_ac.vcf.gz -o 20humans_priors.vcf.gz
```

This output is block compressed and tabix indexed by default, which is necessary for the population priors input. There will be an additional file output called `20humans_priors.vcf.gz.tbi` which is the tabix index file.

The resulting population priors can now be stored in a suitable location to be used for any further runs as required.

```
$ cp 20humans_priors.vcf.gz* /data/population_priors/
```

3.2.2 Task 2 - Run variant calling using population priors

The population priors can now be used to improve variant calling on new members of the population supplying the `--population-priors` parameter to any of the variant caller commands.

```
$ rtg snp -o new_human_snps -t /data/reference/human_reference \
  -I /data/mappings/new_human.txt \
  --population-priors /data/population_priors/20humans_priors.vcf.gz
```

3.3 Somatic variant detection in cancer

Use the following ordered steps to detect somatic variations between normal and tumor samples.

Table : Overview of somatic pipeline tasks.

Task		Command & Utilities	Purpose
1	Format reference data	`rtg format`	Convert reference sequence from FASTA file to RTG Sequence Data Format (SDF)
2	Format read data	`rtg format`	Convert read sequence from FASTA and FASTQ files to RTG Sequence Data Format (SDF)
3	Map reads against the reference genome	`rtg map`	Generate read alignments for the normal and cancer samples, and report in a BAM file for downstream analysis
4	Call somatic variants	`rtg somatic`	Detect somatic variants between the normal and tumor samples

3.3.1 Task 1 - Format reference data (Somatic)

Format the human reference data to RTG SDF using Task 1 of *RTG mapping and sequence variant detection*. In the following tasks it is assumed the human reference SDF is called `grch38`.

3.3.2 Task 2 - Format read data (Somatic)

Format the normal and tumor sample read data sets to RTG SDF using Task 2 of *RTG mapping and sequence variant detection*. In this example we assume there are 20 lanes of data for each sample.

```
$ for ((i=1;i<20;i++)); do
$   rtg format -f fastq -q solexa -o normal_reads_${i} \
    -l /data/reads/normal/${i}/reads_1.fq \
    -r /data/reads/normal/${i}/reads_2.fq \
    --sam-rg "@RG\tID:normal_${lane_id}\tSM:sm_normal\tPL:ILLUMINA"
  done
$ for ((i=1;i<20;i++)); do
$   rtg format -f fastq -q solexa -o tumor_reads_${i} \
    -l /data/reads/tumor/${i}/reads_1.fq \
```

```
    -r /data/reads/tumor/${i}/reads_2.fq \
    --sam-rg "@RG\tID:tumor_${lane_id}\tSM:sm_tumor\tPL:ILLUMINA"
  done
```

3.3.3 Task 3 - Map tumor and normal sample reads against the reference genome

Map the normal and tumor sample reads against the reference genome as in *Task 4 - Map reads to the reference genome*. The mapping must be done with appropriate read group information for each read set with the `--sam-rg` flag. All mappings on the normal should have the same sample ID and all mappings on the tumor should have the same sample ID, but the sample ID should be different between the tumor mappings and normal mappings.

```
$ for ((i=1;i<20;i++)); do
    rtg map -i normal_reads_${i} -t grch38 -o normal_map_${i}
  done
$ for ((i=1;i<20;i++)); do
    rtg map -i cancer_reads_${i} -t grch38 -o cancer_map_${i}
  done
```

3.3.4 Task 4 - Call somatic variants

The `somatic` command is invoked similarly to the `snp` command except you need to specify some extra details. Firstly you need to specify the sample IDs corresponding to the normal and cancer samples, with the `--original` and `--derived` flags respectively. Secondly you may optionally specify the estimated level of contamination of the tumor sample with normal tissue using the `--contamination` flag.

```
$ rtg somatic -t grch38 -o somatic_out --derived sm_tumor --original sm_normal \
  --contamination 0.1 normal_map_*/alignments.bam tumor_map_*/alignments.bam
```

Examining the `snps.vcf.gz` file in the output directory will show a column each for the variants of the normal and tumor samples and will contain variants where the tumor sample differs from the normal sample. The somatic command stores information in the VCF `INFO` fields `NCS`, and `LOH`, and `FORMAT` fields `SSC` and `SS`. For more details about the VCF output file see *Small-variant VCF output file description*.

3.3.5 Using site-specific somatic priors

The `somatic` command has a default prior of 0.000001 (1e-6) for a particular site being somatic. Since the human genome comprises some 3.2 GB, this prior corresponds to an expectation of about 3200 somatic variants in a whole genome sample. Depending on the expected number of variants for a particular sample, it may be appropriate to raise or lower this prior. In general, decreasing the prior increases specificity while increasing the prior increases sensitivity.

Of course, not every site is equally likely to lead to a somatic variant. To support different priors for different sites we provide a facility to set a prior on per site basis via a BED file we call `site-specific somatic priors`. The `--somatic-priors` command line option is used to provide the file to the `somatic` command.

The site-specific somatic priors can cover as many or as few sites as desired. Any site not covered by a specific prior will use the default prior. The format of the file is a BED file where the fourth column of each line gives the explicit prior for the indicated region, for example,

```
1 14906 14910 1e-8
```

denotes that the prior for bases 14907, 14908, 14909, and 14910 on chromosome 1 is 1e-8 rather than the default. (Recall that BED files use 0-based indices.) If the BED file contains more than one prior covering a particular site, then the largest prior covering that site is used. When making a complex call, the prior used is the arithmetic average of priors in the region of the complex call.

A typical starting point for making somatic site-specific priors might include a database of known cancer sites (for example, COSMIC) and a database of sites known to be variant in the population (for example, dbSNP). The idea is that the COSMIC sites are more likely to be somatic and should have a higher prior, while those in dbSNP are less likely to be somatic and should have a lower prior.

The following recipe can be used to build the BED file where some sites have a lower prior of 1e-8 and others have a higher prior of 1e-5. The procedure can be easily modified to incorporate additional inputs each with its own prior.

First, collect prerequisites in the form of VCF files (here using the names `cosmic.vcf.gz` and `dbsnp.vcf.gz`, but, of course, any other VCFs can also be used).

```
$ COSMIC=cosmic.vcf.gz
$ DBSNP=dbsnp.vcf.gz
```

Convert each VCF file into a BED file with the desired priors taking care to convert from 1-based coordinates in VCF to 0-based coordinates in VCF.

```
$ zcat ${COSMIC} | awk -vOFS='\t' '/^[^#]/{print $1,$2-1,$2+length($4)-1,"1e-5"}'\
  | sort -Vu >p0.bed
$ zcat ${DBSNP} | awk -vOFS='\t' '/^[^#]/{print $1,$2-1,$2+length($4)-1,"1e-8"}'\
  | sort -Vu >p1.bed
```

[Optional] Collapse adjacent intervals together. One way of doing this is to use the `bedtools merge` facility. This can result in a smaller final result when the intervals are dense.

```
$ bedtools merge -c 4 -o distinct -i p0.bed >p0.tmp && mv p0.tmp p0.bed
$ bedtools merge -c 4 -o distinct -i p1.bed >p1.tmp && mv p1.tmp p1.bed
```

In general, care must be taken to ensure intersecting sites are handled in the desired manner. Since in this case we want to use COSMIC in preference to dbSNP and prior(COSMIC) > prior(dbSNP), we can simply merge the outputs because the somatic caller will choose the larger prior in the case of overlap.

```
$ sort --merge -V p0.bed p1.bed | rtg bgzip - >somatic-priors.bed.gz
```

To support somatic calling on restricted regions, construct a tabix index for the priors file.

```
$ rtg index -f bed somatic-priors.bed.gz
```

The site-specific somatic BED file is now ready to be used by the `somatic` command:

```
$ rtg somatic --somatic-priors somatic-priors.bed.gz ...
```

3.4 AVR scoring using HAPMAP for model building

AVR (Adaptive Variant Rescoring) is a machine learning technology for learning and predicting which calls are likely correct. It comprises of a learning algorithm that takes training examples and infers a model about what constitutes a good call and a prediction engine which applies the model to variants and estimates their correctness. It includes attributes of the call that are not considered by the internal Bayesian statistics model to make better predictions as to the correctness of a variant call.

Each of the RTG variant callers (`snp`, `family`, `population`) automatically runs a default AVR model, producing an AVR attribute for each sample. The model can be changed with the `--avr-model` parameter, and the AVR functionality can be turned off completely by specifying the special '`none`' model.

Example command line usage. Turn AVR rescoring off:

```
$ rtg family --mother NA12892 --father NA12891 --daughter NA12878 -t grch38 \
  -o trio_variants -I map_trio-bam-files --avr-model none
```

Apply default RTG AVR model:

```
$ rtg family --mother NA12892 --father NA12891 --daughter NA12878 -t grch38 \
  -o trio_variants -I map_trio-bam-files
```

Apply a custom AVR model:

```
$ rtg family --mother NA12892 --father NA12891 --daughter NA12878 -t grch38 \
  -o trio_variants -I map_trio-bam-files --avr-model /path/to/my/custom.avr
```

The effectiveness of AVR is strongly dependent on the quality of the training data. In general, the more training data you have, the better the model will be. Ideally the training data should have the same characteristics as the calls to be predicted; that is, the same platform, the same reference, the same coverage, etc. There also needs to be a balance of positive and negative training examples. In reality, these conditions can only be met to varying degrees, but AVR will try to make the most of what it is given.

A given AVR model is tied to a set of attributes corresponding to fields in VCF records or quantities that are derivable from those fields. The attributes chosen can take into account anomalies associated with different sequencing technologies. Examples of attributes are things like quality of the call, zygosity of the call, strand bias, allele balance, and whether or not the call is complex. Not all attributes are equally predictive and it is the job of the machine learning to determine which combinations of attributes lead to the best predictions. When building a model it is necessary to provide the list of attributes to be used. In general, providing more attributes gives the AVR model a better chance at learning what constitutes a good call. There are two caveats; the attributes used during training need to be present in the calls to be predicted and some attributes like DP are vulnerable to changes in coverage. AVR is able to cope with missing values during both training and prediction.

The training data needs to comprise both positive and negative examples. Ideally we would know exactly the truth status of each training example, but in reality this must be approximated by reference to some combination of baseline information.

In the example that follows, the HAPMAP database will be used to produce and then use an AVR model on a set of output variants, a process that can be used when no appropriate AVR model is already available. The HAPMAP database will be used to determine which of the variants will be considered correct for training purposes. This will introduce two types of error; correct calls which are not in HAPMAP will be marked as negative training examples and a few incorrect calls occurring at HAPMAP sites will be marked as positive training examples.

Data

Reference SDF on which variant calling was performed, in this example assumed to be an existing SDF containing the 1000 genomes build 37 phase 2 reference

(ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_reference_assembly_sequence/hs37d5.fa.gz). For this example this will be called `/data/reference/1000g_v37_phase2`.

The HAPMAP variants file from the Broad Institute data bundle

(ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/2.5/b37/hapmap_3.3.b37.vcf.gz). For this example this will be called `/data/hapmap_3.3.b37.vcf.gz`.

The example will be performed on the merged results of the RTG `family` command for the 1000 genomes CEU trio NA12878, NA12891 and NA12892. For this example this will be called `/data/runs/NA12878trio/family.vcf.gz`.

Table : Overview of basic pipeline tasks.

Task		Command & Utilities	Purpose
1	Create training data	`rtg vcffilter`	To generate positive and negative examples for the AVR machine learning model to train on
2	Build and check AVR model	`rtg avrbuild`, `rtg avrstats`	To create and check an AVR model
3	Use AVR model	`rtg avrpredict`, `rtg snp`, `rtg family`, `rtg population`	To apply the AVR model to the existing output or to use it directly during variant calling
4	Install AVR model	`cp`	Install model in standard RTG model location for later reuse

3.4.1 Task 1 - Create training data

The first step is to use the `vcffilter` command to split the variant calls into positive and negative training examples with respect to HAPMAP. It is also possible to build training sets using the `vcfeval` command, however this is less appropriate for dealing with sites rather than calls, and experiments indicate using `vcffilter` leads to better training sets.

```
$ rtg vcffilter --include-vcf /data/hapmap_3.3.b37.vcf.gz -o pos-NA12878.vcf.gz \
  -i /data/runs/NA12878trio/family.vcf.gz --sample NA12878 --remove-same-as-ref
$ rtg vcffilter --exclude-vcf /data/hapmap_3.3.b37.vcf.gz -o neg-NA12878.vcf.gz \
  -i /data/runs/NA12878trio/family.vcf.gz --sample NA12878 --remove-same-as-ref
```

Optionally check that the training data looks reasonable. There should be a reasonable amount of both positive and negative examples and all expected chromosomes should be represented.

3.4.2 Task 2 - Build and check AVR model

The next step is to build an AVR model. Select the attributes that will be used with some consideration to portability and the nature of the training set. Here we have excluded `XRX` and `LAL` because HAPMAP is primarily SNP locations and does not capture complex calls. By excluding `XRX` and `LAL` we prevent the model from learning that complex calls are bad. We have also excluded `DP` because we want a model somewhat independent of coverage level. Building the model can take a large amount of RAM and several hours. The amount of memory required is proportional to the number of training instances.

A list of derived annotations that can be used are available in the documentation for *avrbuild*. For VCF `INFO` and `FORMAT` annotations check the header of the VCF file for available fields. The VCF fields output by RTG variant callers are described in *Small-variant VCF output file description*.

```
$ rtg avrbuild -o NA12878model.avr --info-annotations DPR \
  --format-annotations DPR,AR,ABP,SBP,RPB,GQ \
  --derived-annotations IC,EP,QD,AN,PD,GQD,ZY \
  --sample NA12878 -p pos-NA12878.vcf.gz -n neg-NA12878.vcf.gz
```

Examine the statistics output to the screen to check things look reasonable. Due to how attributes are computed there can be missing values, but it is a bad sign if any attribute is missing from most samples.

```
Total number of examples: 5073752
Total number of positive examples: 680677
Total number of negative examples: 4393075
Total weight of positive examples: 2536873.27
Total weight of negative examples: 2536876.42
Number of examples with missing values:
DERIVED-AN 0
DERIVED-EP 0
DERIVED-GQD 0
DERIVED-IC 915424
DERIVED-PD 0
```

```
DERIVED-QD 0
DERIVED-ZY 0
FORMAT-ABP 13602
FORMAT-AR 8375
FORMAT-DPR 0
FORMAT-GQ 0
FORMAT-RPB 8375
FORMAT-SBP 37129
INFO-DPR 0
Hold-out score: 69.1821% (96853407/139997752)
Attribute importance estimate for DERIVED-AN: 0.0304% (29443/96853407)
Attribute importance estimate for DERIVED-EP: 1.2691% (1229135/96853407)
Attribute importance estimate for DERIVED-GQD: 4.7844%
(4633873/96853407)
Attribute importance estimate for DERIVED-IC: 4.7748% (4624554/96853407)
Attribute importance estimate for DERIVED-PD: 0.0000% (0/96853407)
Attribute importance estimate for DERIVED-QD: 6.8396% (6624383/96853407)
Attribute importance estimate for DERIVED-ZY: 0.8693% (841925/96853407)
Attribute importance estimate for FORMAT-ABP: 5.4232% (5252533/96853407)
Attribute importance estimate for FORMAT-AR: 2.6008% (2518955/96853407)
Attribute importance estimate for FORMAT-DPR: 3.3186% (3214163/96853407)
Attribute importance estimate for FORMAT-GQ: 3.4268% (3319009/96853407)
Attribute importance estimate for FORMAT-RPB: 0.2176% (210761/96853407)
Attribute importance estimate for FORMAT-SBP: 0.0219% (21196/96853407)
Attribute importance estimate for INFO-DPR: 4.6525% (4506074/96853407)
```

Optionally check the resulting model file using the `avrstats` command. This will produce a short summary of the model including the attributes used during the build.

```
$ rtg avrstats NA12878model.avr
Location : NA12878model.avr
Parameters : avrbuild -o NA12878model.avr --info-annotations DPR --derived-
↪annotations IC,EP,QD,AN,PD,GQD,ZY --format-annotations DPR,AR,ABP,SBP,RPB,GQ --
↪sample NA12878 -p pos-NA12878.vcf.gz -n neg-NA12878.vcf.gz
Date built : 2013-05-17-09-41-17
AVR Version : 1
AVR-ID : 7ded37d7-817f-467b-a7da-73e374719c7f
Type : ML
QUAL used : false
INFO fields : DPR
FORMAT fields : DPR,AR,ABP,SBP,RPB,GQ
Derived fields: IC,EP,QD,AN,PD,GQD,ZY
```

3.4.3 Task 3 - Use AVR model

The model is now ready to be used. It can be applied to the existing variant calling output by using the `avrpredict` command.

```
$ rtg avrpredict --avr-model NA12878model.avr \
  -i /data/runs/NA12878trio/family.vcf.gz -o predict.vcf.gz
```

This will create or update the `AVR` `FORMAT` field in the VCF output file with a score between 0 and 1. The higher the resulting score the more likely it is correct. To select an appropriate cutoff value for further analysis of variants some approaches might include measuring the Ti/Tv ratio or measuring sensitivity against another standard such as OMNI at varying score cutoffs.

The model can also be used directly in any new variant calling runs:

```
$ rtg snp --avr-model NA12878model.avr -t grch38 -o snp_sample_NA12878 --sex␣
↪female \
  -I sample_NA12878-map-files
```

```
$ rtg population --avr-model NA12878model.avr -t grch38 -o pop_variants \
  -I map_population-bam-files
```

3.4.4 Task 4 - Install AVR model

The custom AVR model can be installed into a standard location so that it can be referred to by a short name (rather than the full file path name) in the `avrpredict` and variant caller commands. The default location for AVR models is within a subdirectory of the RTG installation directory called `models`, and each file in that directory with a `.avr` extension is a model that can be accessed by its short name. For example if the `NA12878model.avr` model file is placed in `/path/to/rtg/installation/models/NA12878model.avr` it can be accessed by any user either using the full path to the model:

```
$ rtg snp --avr-model /path/to/rtg/installation/models/NA12878model.avr \
  -o snp_sample_NA12878 -t grch38 --sex female -I sample_NA12878-map-files
```

or, by just the model file name:

```
$ rtg snp --avr-model NA12878model.avr -o snp_sample_NA12878 -t grch38 --sex␣
↪female \
  -I sample_NA12878-map-files
```

The AVR model directory will already contain the models prebuilt by RTG:

- `illumina-exome.avr` - model built from Illumina exome sequencing data. If you are running variant calling Illumina exome data you may want to use this model instead of the default, although the default should still be effective.
- `illumina-wgs.avr` - model built from Illumina whole genome sequencing data. This model is the default model when running normal variant calling.
- **`illumina-somatic.avr` - model built from somatic samples using** Illumina sequencing. It is applicable to somatic variant calling, where a variety of allelic fractions are to be expected in somatic variants. The `somatic` command defaults to this AVR model. If you want to score germline variants in a somatic run, it is preferable to use `illumina-wgs.avr` or `illumina-exome.wgs` instead.
- `alternate.avr` - model built using `XRX`, `ZY` and `GQD` attributes. This should be platform independent and may be a better choice if a more specific model for your data is unavailable. In particular, this model may be more appropriate for scoring the results of variant calling in situations where unusual allele-balance is expected (for example somatic calling with contamination, or calling high amplification data where allele drop out is expected)

It is possible to score a sample with more than one AVR model, by running `avrpredict` with another model and using a different field name specified with `--vcf-score-field`.

3.5 RTG structural variant detection

RTG has developed tools to assist in discovering structural variant regions in whole genome sequencing data. The tools can be used to locate likely structural variant breakpoints and regions that have been duplicated or deleted. These tools are capable of processing whole genome mapping data containing multiple read groups in a streamlined fashion.

In this example, it is assumed that alignment has been carried out as described in *Task 4 - Map reads to the reference genome*. For structural variant detection it is particularly important to specify the read group information with the `--sam-rg` flag either during the formatting of the reads or for the map command explicitly. The structural variant tools currently requires the `PL` (platform) attribute to be either `ILLUMINA` (for Illumina reads) or `COMPLETE` (for Complete Genomics reads).

Table : Overview of structural variants analysis pipeline tasks.

Task		Command & Utilities	Purpose
1	Prepare read group statistics file	`find`	Identify read group statistics files created during mapping
2	Find structural variants with sv	`rtg sv`	Process prepared mapping results to identify likely structural variants
3	Find structural variants with discord	`rtg discord`	Process prepared mapping results to identify likely structural variant breakends
4	Find copy number variants	`rtg cnv`	Detect copy number variants between a pair of samples

3.5.1 Task 1 - Prepare Read-group statistics files

Mapping identifies discordant read matings and inserts the pair information for unique unmated reads into the SAM records. The `map` command also produces a file within the directory called `rgstats.tsv` containing read group statistics.

The `sv` and `discord` structural variant callers require the read group statistics files to be supplied, so as with multiple BAM files, it is possible to create a text file listing the locations of all the required statistics files:

```
$ find /partition2/map_sample_NA12878 -name "rgstats.tsv" \
  > sample_NA12878-rgstats-files
```

3.5.2 Task 2 - Find structural variants with sv

Once mapping is complete one can run the structural variants analysis tool. To run the `sv` tool you need to supply the mapping BAM files and the read group statistics files. As with the variant caller commands, this can be a large number of files and so input can be specified using list files.

```
$ rtg sv -t grch38 -o sample_NA12878-sv -I sample_NA12878-bam-files \
  -R sample_NA12878-rgstats-files
$ ls -l sample_NA12878-sv
    30 Oct 1 16:56 done
 15440 Oct 1 16:56 progress
     0 Oct 1 16:56 summary.txt
122762 Oct 1 16:56 sv_bayesian.tsv.gz
  9032 Oct 1 16:56 sv.log
```

The file `sv_bayesian.tsv.gz` contains a trace of the strengths of alternative bayesian hypothesis at points along the reference genome. The currently supported hypotheses are shown in the following table.

Table : Structural variant hypotheses.

Hypothesis	Semantics
`normal`	Normal mappings, no structural variants
`duplicate`	Above normal mappings, potential duplication region
`delete`	Below normal mappings, potential deletion region
`delete-left`	Mapping data suggest the left breakpoint of a deletion
`delete-right`	Mapping data suggest the right breakpoint of a deletion
`duplicate-left`	Mapping data suggest the left boundary of a region that has been copied elsewhere
`duplicate-right`	Mapping data suggest the right boundary of a region that has been copied elsewhere
`breakpoint`	Mapping data suggest this location has received an insertion of copied genome
`novel-insertion`	Mapping data suggests this location has received an insertion of material not present in the reference

For convenience, the last column of the output file gives the index of the hypothesis with the maximum strength, to make it easier to identify regions where this changes for further investigation by the researcher.

The `sv` command also supports calling on individual chromosomes (or regions within a chromosome) with the `--region` parameter, and this can be used to increase overall throughput.

```
$ for seq in M {1..22} X Y; do
    rtg sv -o sv_sample_NA12878-chr${seq} -t grch38 -I sample_NA12878-sam-files \
    -R sample_NA12878-rgstats-files --region chr${seq}
  done
```

3.5.3 Task 3 - Find structural variants with discord

A second tool for finding structural variant break-ends is based on detecting cluster of discordantly mapped reads, those where both ends of a read are mapped but are mapped either in an unexpected orientation or with a `TLEN` outside the normal range for that read group. As with the `sv` command, `discord` requires the read group statistics to be supplied.

```
$ rtg discord -t grch38 -o discord_sample_NA12878 -I sample_NA12878-bam-files \
  -R sample_NA12878-rgstats-files --bed
```

As with `sv`, the `discord` command also supports using the `--region` flag:

```
$ for seq in M {1..22} X Y; do
    rtg discord -o discord_sample_NA12878-chr${seq} -t grch38 \
    -I sample_NA12878-bam-files -R sample_NA12878-rgstats-files \
    --region chr${seq}
  done
```

The default output is in VCF format, following the VCF 4.1 specification for break-ends. However, as most third-party tools currently don't support this type of VCF, it is also possible to output each break-end as a separate region in a BED file.

3.5.4 Task 4 - Report copy number variation statistics

With two genome samples, one can compare the relative depth of coverage by region to identify copy number variation ratios that may indicate structural variation. A common use case is where you have two samples from a cancer patient, one from normal tissue and another from a tumor.

Run the `cnv` command with the default bucket size of 100, this is the number of nucleotides for which to average the coverage.

```
$ rtg cnv -o cnv_s1_s2 -I sample1-map-files -J sample2-map-files
```

View the resulting output as a set of records that show cnv ratio at locations across the genome, where the locations are defined by the bucket size.

```
$ zless cnv_s1_/cnv.txt.gz
```

For deeper investigation, contact Real Time Genomics technical support for extensible reporting scripts specific to copy number variation reporting.

3.6 Ion Torrent bacterial mapping and sequence variant detection

The following example supports the steps typical to bacterial genome analysis in which an Ion Torrent sequencer has generated reads at 10 fold coverage.

Data

The baseline uses actual data downloaded from the Ion community. The reference sequence is "Escherichia coli K-12 sub-strain DH10B", available from the NCBI RefSeq database, accession NC_010473 (http://www.ncbi.

nlm.nih.gov/nuccore/NC_010473). The read data is comprised of Ion Torrent PGM run B14-387, which can be found at the Ion community (http://lifetech-it.hosted.jivesoftware.com, requires registration).

Table : Overview of basic pipeline tasks.

Task		Command & Utilities	Purpose
1	Format reference data	`rtg format`	Convert reference sequence from FASTA file to RTG Sequence Data Format (SDF)
2	Format read data	`rtg format`	Convert read sequence from FASTA and FASTQ files to RTG Sequence Data Format (SDF)
3	Map reads against the reference genome	`rtg map`	Generate read alignments for the normal and cancer samples, and report in a BAM file for downstream analysis
4	Call sequence variants in haploid mode	`rtg snp`	Detect SNPs, MNPs, and indels in haploid sample relative to the reference genome

3.6.1 Task 1 - Format bacterial reference data (Ion Torrent)

Mapping and variant detection requires a conversion of the reference genome from FASTA files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory containing the reference genome.

Use the `format` command to convert the FASTA file into an SDF directory for the reference genome.

```
$ rtg format -o ecoli-DH10B \
  /data/bacteria/Escherichia_coli_K_12_substr__DH10B_uid58979/NC_010473.fna
```

3.6.2 Task 2 - Format read data (Ion Torrent)

Mapping and variant detection of Ion Torrent data requires a conversion of the read sequence data from FASTQ files into the RTG SDF format. Additionally, it is recommended that read trimming based on the quality data present within the FASTQ file be performed as part of this conversion.

Use the `format` command to convert the read FASTQ file into an SDF directory, using the quality threshold option to trim poor quality ends of reads.

```
$ rtg format -f fastq -q solexa --trim-threshold=15 -o B14-387-reads \
  /data/reads/R_2011_07_19_20_05_38_user_B14-387-r121336-314_pool30-ms_Auto_B14\
-387-r121336-314_pool30-ms_8399.fastq
```

3.6.3 Task 3 - Map Ion Torrent reads to the reference genome

Map the sequence reads against the reference genome to generate alignments in BAM format.

The RTG `map` command provides a means for the mapping of Ion Torrent reads to a reference genome. When mapping Ion Torrent reads, a read group with the platform field (`PL`) set to `IONTORRENT` should be provided.

```
$ rtg map -i B14-387-reads -t ecoli-DH10B -o B14-387-map \
  --sam-rg "@RG\tID:B14-387-r121336\tSM:B14-387\tPL:IONTORRENT"
```

Multiple files are written to the output directory of the mapping run. For further variant calling, the `alignments.bam` file has the associated required index file `alignments.bam.bai`. The additional files `alignments.bam.calibration` contains metadata to provide more accurate variant calling.

3.6.4 Task 4 - Call sequence variants in haploid mode

Call haploid sequence variants in the mapped reads against the reference genome to generate a variants file in the VCF format.

The `snp` command will automatically set machine error calibration parameters according to the platform (PL attribute) specified in the SAM read group, in this example to the Ion Torrent parameters. The `snp` command defaults to diploid variant calling, so for this bacterial example haploid mode will be specified. The automatically included `.calibration` files provide additional information specific to the mapping data for improved variant calling.

```
$ rtg snp -t ecoli-DH10B -o B14-387-snp --ploidy=haploid B14-387-map/alignments.bam
```

Examining the `snps.vcf.gz` file in the output directory will show that variants have been called in haploid mode. For more details about the VCF output file see *Small-variant VCF output file description*.

3.7 RTG contaminant filtering

Use the following set of tasks to remove contaminated reads from a sequenced DNA sample.

The RTG contamination filter, called `mapf`, removes contaminant reads by mapping against a database of potential contaminant sequences. For example, a bacterial metagenomic sample may have some amount of human sequence contaminating it. The following process removes any human reads leaving only bacteria reads.

Table : Overview of contaminant filtering tasks.

Task		Command & Utilities	Purpose
1	Format reference data	`rtg format`	Convert reference sequence from FASTA file to RTG Sequence Data Format (SDF)
2	Format read data	`rtg format`	Convert read sequence from FASTA and FASTQ files to RTG Sequence Data Format (SDF)
3	Run contamination filter	`rtg mapf`	Produce the SDF file of reads which map to the contaminant and the SDF file of those that do not
4	Manage filtered reads	`mv`	Set up the results for use in further processing

3.7.1 Task 1 - Format reference data (contaminant filtering)

RTG tools require a conversion of reference sequences from FASTA files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory containing the reference genome.

First, observe a typical genome reference with multiple chromosome sequences stored in compressed FASTA format.

```
$ ls -l /data/human/hg19/
43026389 Mar 21 2009 chr10.fa.gz
42966674 Mar 21 2009 chr11.fa.gz
42648875 Mar 21 2009 chr12.fa.gz
31517348 Mar 21 2009 chr13.fa.gz
28970334 Mar 21 2009 chr14.fa.gz
26828094 Mar 21 2009 chr15.fa.gz
25667827 Mar 21 2009 chr16.fa.gz
25139792 Mar 21 2009 chr17.fa.gz
24574571 Mar 21 2009 chr18.fa.gz
17606811 Mar 21 2009 chr19.fa.gz
73773666 Mar 21 2009 chr1.fa.gz
```

```
19513342 Mar 21 2009 chr20.fa.gz
11549785 Mar 21 2009 chr21.fa.gz
11327826 Mar 21 2009 chr22.fa.gz
78240395 Mar 21 2009 chr2.fa.gz
64033758 Mar 21 2009 chr3.fa.gz
61700369 Mar 21 2009 chr4.fa.gz
58378199 Mar 21 2009 chr5.fa.gz
54997756 Mar 21 2009 chr6.fa.gz
50667196 Mar 21 2009 chr7.fa.gz
46889258 Mar 21 2009 chr8.fa.gz
39464200 Mar 21 2009 chr9.fa.gz
    5537 Mar 21 2009 chrM.fa.gz
49278128 Mar 21 2009 chrX.fa.gz
 8276338 Mar 21 2009 chrY.fa.gz
```

Now, use the `format` command to convert multiple input files into a single SDF directory for the reference.

```
$ rtg format -o hg19 /data/human/hg19/chrM.fa.gz \
  /data/human/hg19/chr[1-9].fa.gz \
  /data/human/hg19/chr1[0-9].fa.gz \
  /data/human/hg19/chr2[0-9].fa.gz \
  /data/human/hg19/chrX.fa.gz \
  /data/human/hg19/chrY.fa.gz
```

This takes the human reference FASTA files and creates a directory called `hg19` containing the SDF, with chromosomes ordered in the standard UCSC ordering. You can use the `sdfstats` command to show statistics for your reference SDF, including the individual sequence lengths.

```
$ rtg sdfstats --lengths hg19
Type : DNA
Number of sequences: 25
Maximum length : 249250621
Minimum length : 16571
Sequence names : yes
N : 234350281
A : 844868045
C : 585017944
G : 585360436
T : 846097277
Total residues : 3095693983
Residue qualities : no
Sequence lengths:
chrM 16571
chr1 249250621
chr2 243199373
chr3 198022430
chr4 191154276
chr5 180915260
chr6 171115067
chr7 159138663
chr8 146364022
chr9 141213431
chr10 135534747
chr11 135006516
chr12 133851895
chr13 115169878
chr14 107349540
chr15 102531392
chr16 90354753
chr17 81195210
chr18 78077248
chr19 59128983
```

```
chr20 63025520
chr21 48129895
chr22 51304566
chrX 155270560
chrY 59373566
```

3.7.2 Task 2 - Format read data (contaminant filtering)

RTG tools require a conversion of read sequence data from FASTA or FASTQ files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory for the sample reads.

Take a paired set of reads in FASTQ format and convert it into RTG data format (SDF). This example shows one run of data, taking as input both left and right mate pairs from the same run.

```
$ rtg format -f fastq -q sanger -o bacteria-sample \
  -l /data/reads/bacteria/sample_1.fq \
  -r /data/reads/bacteria/sample_2.fq
```

This creates a directory named bacteria-sample with two subdirectories, named left and right. Use the `sdfstats` command to verify this step.

```
$ rtg sdfstats bacteria-sample
```

3.7.3 Task 3 - Run contamination filter

The `mapf` command functions in much the same way as the `map` command, but instead of producing BAM files of the mappings it produces two SDF directories, one containing reads that map to the reference and the other with reads that do not map. As with the `map` command there are multiple tuning parameters to adjust sensitivity and selectivity of the mappings. As with the `map` command, you can use the `--start-read` and `--end-read` flags to perform the mapping in smaller sections if required. The default `mapf` settings are similar to `map` although the word size and step sizes have been adjusted to yield more sensitive mappings.

```
$ rtg mapf -t hg19 -i bacteria-sample -o filter-sample
```

In the `filter-sample` output directory there are, amongst other files, two directories named `alignments.sdf` and `unmapped.sdf`. The `alignments.sdf` directory is an SDF of the reads that mapped to the reference, and the `unmapped.sdf` directory is an SDF of the reads that did not map.

```
$ ls -l filter-sample/
   4096 Sep 30 16:02 alignments.sdf/
     33 Sep 30 16:02 done
2776886 Sep 30 16:02 mapf.log
  12625 Sep 30 16:02 progress
    143 Sep 30 16:02 summary.txt
   4096 Sep 30 16:02 unmapped.sdf/
```

3.7.4 Task 4 - Manage filtered reads

Depending on the use case, either rename, move or delete the filtered SDF directories as necessary. In this example the reads that did not map to the contamination reference are to be used in further processing, so rename the `unmapped.sdf` directory.

```
$ mv filter-sample/unmapped.sdf bacteria-sample-filtered
```

The filtered read set is now ready for subsequent processing, such as with the `mapx` or `species` tools.

3.8 RTG translated protein searching

Use the following set of tasks to search DNA reads against a protein data set.

The RTG protein search tool, `mapx` translates nucleotide reads into protein space and search them against a protein data set. For example, a sample taken from a human gut can be searched against a protein data set to determine which kinds of protein families are present in the sample.

For this example we will search a human gut sample read set against an NCBI non-redundant protein data set. In the following tasks it is assumed non-redundant protein data set is called `nr.fasta` and the human gut sample is called `human-gut.fastq`.

Table : Overview of translated protein searching tasks.

Task		Command & Utilities	Purpose
1	Format protein data set	`rtg format`	Convert protein data set from FASTA to RTG sequence data format (SDF)
2	Format DNA read set	`rtg format`	Convert read sequence from FASTA and FASTQ files to RTG Sequence Data Format (SDF)
3	Search against protein data set	`rtg mapx`	Generate search results with alignments in tabular format

3.8.1 Task 1 - Format protein data set

The `mapx` command requires a conversion of a protein data set from FASTA files into RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory containing the protein data set.

```
$ rtg format -p -o nr /data/NCBI-nr/nr.fasta
```

The above command will take the `nr.fasta` file and create a directory called `nr` containing the SDF. Note that the `-p` option is used to create the SDF with protein data.

3.8.2 Task 2 - Format DNA read set

The `mapx` command requires a conversion of the DNA read set data from FASTA or FASTQ files into RTG SDF format. This task will be completed with the `format` command. The following command assumes the sample read data set is in Solexa FASTQ format.

```
$ rtg format -f fastq -q solexa -o human-gut /data/human-gut-sample.fastq
```

3.8.3 Task 3 - Search against protein data set

Search the DNA reads against the protein data set and generate alignments in tabular format.

The `mapx` command provides multiple tuning parameters to adjust sensitivity and selectivity at the search stage. As with the `map` command, you can use the `--start-read` and `--end-read` flags to perform the mapping in smaller sections if required. In general, protein search strategies are based on protein similarity also known as identity.

The search example below uses a sensitivity setting that will guarantee reporting with reads that align with 4 substitutions and 1 indels.

```
$ rtg mapx -t nr -i human-gut -o mapx_results -a 3 -b 1
```

The `alignments.tsv.gz` file in the `mapx_results` output directory contains tabular output with alignments. For more information about this output format see *Mapx output file description*.

3.9 RTG species frequency estimation

Use the following set of tasks to estimate the frequency of bacterial species in a metagenomic sample. The RTG `species` frequency estimator, called species, takes a set of reads mapped against a bacterial database and from this estimates the relative frequency of each species in the database.

Table : Overview of species frequency estimation tasks.

Task		Command & Utilities	Purpose
1	Format reference data	`rtg format`	Convert reference sequence from FASTA file to RTG Sequence Data Format (SDF)
2	Format read data	`rtg format`	Convert read sequence from FASTA and FASTQ files to RTG Sequence Data Format (SDF)
3	Run contamination filter (optional)	`rtg mapf`	Produce the SDF file of reads which map to the contaminant and the SDF file of those that do not
4	Map metagenomic reads against bacterial database	`rtg map`	Generate read alignments against a given reference, and report in a BAM file for downstream analysis
5	Run species estimator	`rtg species`	Produce a text file which contains a list of species, one per line, with an estimate of the relative frequency in the sample

3.9.1 Task 1 - Format reference data (species)

RTG tools require a conversion of reference sequences from FASTA files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory containing the reference sequences.

Use the `format` command to convert multiple input files into a single SDF directory for the reference database.

```
$ rtg format -o bacteria-db /data/bacteria/db/*.fa.gz
```

This takes the reference FASTA files and creates a directory called `bacteria-db` containing the SDF. You can use the `sdfstats` command to show statistics for your reference SDF.

```
$ rtg sdfstats bacteria-db
Type               : DNA
Number of sequences: 311276
Maximum length     : 13033779
Minimum length     : 0
Sequence names     : yes
N                  : 33864547
A                  : 4167856151
C                  : 4080877385
G                  : 4072353906
T                  : 4177108579
Total residues     : 16532060568
Residue qualities  : no
```

Alternatively a species reference SDF for running the `species` command can be obtained from our website (http://www.realtimegenomics.com).

3.9.2 Task 2 - Format read data (species)

RTG tools require a conversion of read sequence data from FASTA or FASTQ files into the RTG SDF format. This task will be completed with the `format` command. The conversion will create an SDF directory for the sample reads.

Take a paired set of reads in FASTQ format and convert it into RTG data format (SDF). This example shows one run of data, taking as input both left and right mate pairs from the same run.

```
$ rtg format -f fastq -q sanger -o bacteria-sample \
  -l /data/reads/bacteria/sample_1.fq \
  -r /data/reads/bacteria/sample_2.fq
```

This creates a directory named `bacteria-sample` with two subdirectories, named 'left' and 'right'. Use the `sdfstats` command to verify this step.

```
$ rtg sdfstats bacteria-sample
```

3.9.3 Task 3 - Run contamination filter (optional)

Optionally filter the metagenomic read sample to remove human contamination using Tasks 1 through 4 of *RTG contaminant filtering*

3.9.4 Task 4 - Map metagenomic reads against bacterial database

Map the metagenomic reads against the reference database to generate alignments in the BAM format (Binary Sequence Alignment/Map file format). The read set in this example is paired end.

It is recommended that during mapping either the `--max-top-results` flag be set to a high value, such as 100, or that the `--all-hits` option be used. This helps ensure that all relevant species in the database are accurately represented in the output. However, note that a very large `--max-top-results` requires additional memory during mapping.

```
$ rtg map -i bacteria-sample -t bacteria-db -o map-sample -n 100
```

3.9.5 Task 5 - Run species estimator

The species estimator, `species`, takes as input the BAM format files from the mapping performed against the reference database.

```
$ rtg species -t bacteria-db -o species-result map-sample/alignments.bam
```

This run generates a new output directory `species_result`. The main result file in this directory will be called `species.tsv`. In the output the bacterial species are ordered from most to least abundant. The output file can be directly loaded into a spreadsheet program like Microsoft Excel.

The `species.tsv` file contains results for both species with associated genomic sequences and internal nodes in the taxonomy. In some scenarios it will only be necessary to examine those rows corresponding to sequences in the database, such rows have a `Y` in the `has-sequence` column. Internal taxonomy nodes (i.e. ones that have no associated sequence data) always have a breadth and depth of coverage of zero because no reads directly map to them. For further detail on the `species.tsv` file format see *Species results file description*

Also produced is an HTML5 summary file called `index.html` which contains an interactive pie chart detailing the results.

The best results are obtained when as many relevant records as possible are given to the species estimator. If you have insufficient memory to use all your mapping results then using the filtering options may help. You could filter the results by selecting mappings with good alignment scores or mated only reads.

3.10 RTG sample similarity

Use the following set of tasks to produce a similarity matrix from the comparison of a group of read sets. An example use case is in metagenomics where several bacteria samples taken from different sites need to be compared.

The `similarity` command performs a similarity analysis on multiple read sets independent of any reference genome. It does this by examining *k*-mer word frequencies and the intersections between sets of reads.

Table : Overview of sample similarity tasks.

Task		Command & Utilities	Purpose
1	Prepare read sets	`rtg format`	Convert reference sequence from FASTA file to RTG Sequence Data Format (SDF)
2	Generate read set name map	`text-editor`	Produce the map of names to read set SDF locations
3	Run similarity tool	`rtg similarity`	Process the read sets for similarity

3.10.1 Task 1 - Prepare read sets

RTG tools require a conversion of read sequence data from FASTA or FASTQ files into the RTG SDF format. This task will be completed with the format command. The conversion will create an SDF directory for the sample reads.

Take a paired set of reads in FASTQ format and convert it into RTG data format (SDF). This example shows one run of data, taking as input both left and right mate pairs from the same run.

```
$ rtg format -f fastq -q sanger -o /data/reads/read-sample1-sdf \
  -l /data/reads/fastq/read-sample1_1.fq \
  -r /data/reads/fastq/read-sample2_2.fq
```

This creates a directory named 'read-sample1-sdf' with two subdirectories, named 'left' and 'right'. Use the `sdfstats` command to verify this step.

```
$ rtg sdfstats /data/reads/read-sample1-sdf
```

Repeat for all read samples to be compared. This example shows how this can be done with the format command in a loop.

```
$ for left_fq in /data/reads/fastq/*_1.fq; do
    right_fq=${left_fq/_1.fq/_2.fq}
    sample_id=$(basename ${left_fq/_1.fq})
    rtg format -f fastq -q sanger -o /data/reads/${sample_id}-sdf -l ${left_fq} \
      -r ${right_fq}
  done
```

3.10.2 Task 2 - Generate read set name map

With a text editor, or other tool, create a text file containing a list of sample name to sample read SDF file locations. If two or more read sets are from the same sample they can be combined by giving them the same sample name in the file list.

```
$ cat read-set-list.txt
sample1 /data/reads/read-sample1-sdf
sample2 /data/reads/read-sample2-sdf
sample3 /data/reads/read-sample3-sdf
sample4 /data/reads/read-sample4-sdf
sample5 /data/reads/read-sample5-sdf
```

3.10.3 Task 3 - Run similarity tool

Run the `similarity` command setting the *k*-mer word size (`-w` parameter) and the step size (`-s` parameter) on the read sets by specifying the file listing the read sets. Some experimentation should be performed with different word and step size parameters to find good trade-offs between memory usage and run time. Should it be necessary to reduce the memory used it is possible to limit the number of reads used from each SDF by specifying the `--max-reads` parameter.

```
$ rtg similarity -w 25 -s 25 --max-reads 1000000 -I read-set-list.txt \
  -o similarity-output
```

The program puts its output in the specified output directory.

```
$ ls similarity-output/
 4693 Aug 29 20:17 closest.tre
19393 Aug 29 20:17 closest.xml
   33 Aug 29 20:17 done
11363 Aug 29 20:17 similarity.log
48901 Aug 29 20:17 similarity.tsv
  693 Aug 29 20:17 progress
```

The `similarity.tsv` file is a tab separated file containing a matrix of counts of the number of *k*-mers in common between each pair of samples. The `closest.tre` and `closest.xml` files are nearest neighbor trees built from the counts from the similarity matrix. The `closest.tre` is in Newick format and the `closest.xml` file is phyloXML. The `similarity.pca` file contains a principal component analysis on the similarity matrix in `similarity.tsv`.

You may wish to view `closest.tre` or `closest.xml` in your preferred tree viewing tool or use the principal component analysis output in `similarity.pca` to produce a three-dimensional grouping plot showing visually the clustering of samples.

CHAPTER

FOUR

ADMINISTRATION & CAPACITY PLANNING

4.1 Advanced installation configuration

RTG software can be shared by a group of users by installing on a centrally available file directory or shared drive. Assignment of execution privileges can be determined by the administrator, independent of the software license file. For commercial users, the software license prepared by Real Time Genomics (`rtg-license.txt`) need only be included in the same directory as the executable (`RTG.jar`) and the run-time scripts (`rtg` or `rtg.bat`).

During installation on Unix systems, a configuration file named `rtg.cfg` is created in the installation directory. By editing this configuration file, one may alter further configuration variables appropriate to the specific deployment requirements of the organization. On Windows systems, these variables are set in the `rtg.bat` file in the installation directory. These configuration variables include:

Variable	Description
`RTG_MEM`	Specify the maximum memory for Java run-time execution. Use a G suffix for gigabytes, e.g.: `RTG_MEM=48G`. The default memory allocation is 90% of system memory.
`RTG_JAVA`	Specify the path to Java (default assumes current path).
`RTG_JAR`	Indicate the path to the `RTG.jar` executable (default assumes current path).
`RTG_JAVA_OPTS`	Provide any additional Java JVM options.
`RTG_DEFAULT_THREADS`	By default any RTG module with a `--threads` parameter will automatically use the number of cores as the number of threads. This setting makes the specified number the default for the `--threads` parameter instead.
`RTG_PROXY`	Specify the http proxy server for TalkBack exception management (default is no http proxy).
`RTG_TALKBACK`	Send log files for crash-severity exception conditions (default is true, set to false to disable).
`RTG_USAGE`	If set to true, enable simple usage logging.
`RTG_USAGE_DIR`	Destination directory when performing single-user file-based usage logging.
`RTG_USAGE_HOST`	Server URL when performing server-based logging.
`RTG_USAGE_OPTIONAL`	May contain a comma-separated list of the names of optional fields to include in usage logging (when enabled). Any of `username`, `hostname` and `commandline` may be set here.
`RTG_REFERENCES_DIR`	Specifies an alternate directory containing metagenomic pipeline reference datasets.
`RTG_MODELS_DIR`	Specifies an alternate directory containing AVR models.

4.2 Run-time performance optimization

CPU — Multi-core operation finishes jobs faster by processing multiple application threads in parallel. By default RTG uses all available cores of a multi-processor server node. With a command line parameter setting, RTG operation can be limited to a specified number of cores if desired.

Memory — Adding more memory can improve performance where very high read coverage is desired. RTG creates and uses indexes to speed up genomic data processing. The more RAM you have, the more reads you can

process in memory in a run. We use 48 GB as a rule of thumb for processing human data. However, a smaller number of reads can be processed in as little as 2 GB.

Disk Capacity — Disk requirements are highly dependent on the size of the underlying data sets, the amount of information needed to hold quality scores, and the number of runs needed to investigate the impact of varying levels of sensitivity. Though all data is handled and stored in compressed form by default, a realistic minimum disk size for handling human data is 1 TB. As a rule of thumb, for every 2 GB of input read data expect to add 1 GB of index data and 1 GB of output files per run. Additionally, leave another 2 GB free for temporary storage during processing.

4.3 Alternate configurations

Demonstration system — For training, testing, demonstrating, processing and otherwise working with smaller genomes, RTG works just fine on a newer laptop system with an Intel processor. For example, product testing in support of this documentation was executed on a MacBook PC (Intel Core 2 Duo processor, 2.1 GHz clock speed, 1 processor, 2 cores, 3 MB L2 Cache, 4 GB RAM, 290 GB 5400 RPM Serial-ATA disk)

Clustered system — The comparison of genomic variation on a large scale demands extensive processing capability. Assuming standard CPU hardware as described above, scale up to meet your institutional or major product needs by adding more rack-mounted boards and blades into rack servers in your data center. To estimate the number of cores required, first estimate the number of jobs to be run, noting size and sensitivity requirements. Then apply the appropriate benchmark figures for different size jobs run with varying sensitivity, dividing the number of reads to be processed by the reads/second/core.

4.4 Exception management - TalkBack and log file

Many RTG commands generate a log file with each run that is saved to the results output directory. The contents of the file contain lists of job parameters, system configuration, and run-time information.

In the case of internal exceptions, additional information is recorded in the log file specific to the problem encountered. Fatal exceptions are trapped and notification is sent to Real Time Genomics with a copy of the log file. This mechanism is called TalkBack and uses an embedded URL to which RTG sends the report.

The following sample log displays the software version information, parameter list, and run-time progress.

```
2009-09-05 21:38:10 RTG version = v2.0b build 20013 (2009-10-03)
2009-09-05 21:38:10 java.runtime.name = Java(TM) SE Runtime Environment
2009-09-05 21:38:10 java.runtime.version = 1.6.0_07-b06-153
2009-09-05 21:38:10 os.arch = x86_64
2009-09-05 21:38:10 os.freememory = 1792544768
2009-09-05 21:38:10 os.name = Mac OS X
2009-09-05 21:38:10 os.totalmemory = 4294967296
2009-09-05 21:38:10 os.version = 10.5.8
2009-09-05 21:38:10 Command line arguments: [-a, 1, -b, 0, -w, 16, -f, topn, -n, 5,
↪ -P, -o, pflow, -i, pfreads, -t, pftemplate]
2009-09-05 21:38:10 NgsParams threshold=20 threads=2
2009-09-05 21:39:59  Index[0] memory performance
```

TalkBack may be disabled by adding `RTG_TALK_BACK=false` to the `rtg.cfg` configuration file (Unix) or the `rtg.bat` file (Window) as described in *Advanced installation configuration*.

4.5 Usage logging

RTG has the ability to record simple command usage information for submission to Real Time Genomics. The first time RTG is run (typically during installation), the user will be asked whether to enable usage logging. This information may be required for customers with a pay-per-use license. Other customers may choose to send this

information to give Real Time Genomics feedback on which commands and features are commonly used or to locally log RTG command use for their own analysis.

A usage record contains the following fields:

- Time and date
- License serial number
- Unique ID for the run
- Version of RTG software
- RTG command name, without parameters (e.g. map)
- Status (Started / Failed / Succeeded)
- A command-specific field (e.g. number of reads)

For example:

```
2013-02-11 11:38:38007   4f6c2eca-0bfc-4267-be70-b7baa85ebf66    RTG Core v2.7␣
↪build d74f45d (2013-02-04)   format  Start   N/A
```

No confidential information is included in these records. It is possible to add extra fields, such as the user name running the command, host name of the machine running the command, and full command-line parameters, however as these fields may contain confidential information, they must be explicitly enabled as described in *Advanced installation configuration*.

When RTG is first installed, you will be asked whether to enable user logging. Usage logging can also be manually enabled by editing the `rtg.cfg` file (or `rtg.bat` file on Windows) and setting `RTG_USAGE=true`. If the `RTG_USAGE_DIR` and `RTG_USAGE_HOST` settings are empty, the default behavior is to directly submit usage records to an RTG hosted server via HTTPS. This feature requires the machine running RTG to have access to the Internet.

For cases where the machines running RTG do not have access to the Internet, there are two alternatives for collecting usage information.

4.5.1 Single-user, single machine

Usage information can be recorded directly to a text file. To enable this option, edit the `rtg.cfg` file (or `rtg.bat` file on Windows), and set the `RTG_USAGE_DIR` to the name of a directory where the user has write permissions. For example:

```
RTG_USAGE=true
RTG_USAGE_DIR=/opt/rtg-usage
```

Within this directory, the RTG usage information will be written to a text file named after the date of the current month, in the form `YYYY-MM.txt`. A new file will be created each month. This text file can be manually sent to Real Time Genomics when requested.

4.5.2 Multi-user or multiple machines

In this case, a local server can be started to collect usage information from compute nodes and recorded to local files for later manual submission. To configure this method of collecting usage information, edit the `rtg.cfg` file (or `rtg.bat` file on Windows), and set the `RTG_USAGE_DIR` to the name of a directory where the local server will store usage logs, and `RTG_USAGE_HOST` to a URL consisting of the name of the local machine that will run the server and the network port on which the server will listen. For example if the server will be run on a machine named `gridhost.mylan.net`, listening on port `9090`, writing usage information into the directory `/opt/rtg-usage/`, set:

```
RTG_USAGE=true
RTG_USAGE_DIR=/opt/rtg-usage
RTG_USAGE_HOST=http://gridhost.mylan.net:9090/
```

On the machine `gridhost`, run the command:

$ rtg usageserver

Which will start the local usage server listening. Now when RTG commands are run on other nodes or as other users, they will submit usage records to this sever for collation.

Within the usage directory, the RTG usage information will be written to a text file named after the date of the current month, in the form `YYYY-MM.txt`. A new file will be created each month. This text file can be manually sent to Real Time Genomics when requested.

4.5.3 Advanced usage configuration

If you wish to augment usage information with any of the optional fields, edit the `rtg.cfg` file (or `rtg.bat` file on Windows) and set the `RTG_USAGE_OPTIONAL` to a comma separated list containing any of the following:

- `username` - adds the username of the user running the RTG command.
- `hostname` - adds the machine name running the RTG command.
- `commandline` - adds the command line, including parameters, of the RTG command (this field will be truncated if the length exceeds 1000 characters).

For example:

```
RTG_USAGE_OPTIONAL=username,hostname,commandline
```

4.6 Command-line color highlighting

Some RTG commands make use of ANSI colors to visually enhance terminal output, and the decision as to whether to colorize the output is automatically determined, although some commands also contain additional flags to control colorization.

The default behaviour of output colorization can be configured by defining a Java system property named `rtg.default-markup` with an appropriate value and supplying it via RTG_JAVA_OPTS. For example, to disable output colorization, use:

```
RTG_JAVA_OPTS="-Drtg.default-markup=none"
```

The possible values for `rtg.default-markup` are:

- `auto` - automatically enable ANSI markup when running on non-Windows OS and when I/O is detected to be a console.
- `none` - disable ANSI markup.
- `ansi` - enable ANSI markup. This may be useful if you are using Windows OS and have installed an ANSI-capable terminal such as ANSICON, ConEmu or Console 2.

CHAPTER

FIVE

APPENDIX

5.1 RTG gapped alignment technical description

Real Time Genomics utilizes its own DNA sequence alignment tool and scoring system for aligned reads. Most methods for sequence comparison and alignment use a small set of operations derived from the notion of *edit distance*[1] to discover differences between two DNA sequences. The edit operations introduce insertions, deletions, and substitutions to transform one sequence into another. Alignments are termed global if they extend over all residues of both sequences.

Most programs for finding global alignments are based on the Needleman-Wunsch algorithm[2]. Alternatively, alignments may be local, in which case reported alignments may contain subsequences of the input sequences. The Smith-Waterman variation on the Needleman-Wunsch algorithm finds such alignments[3]. The proprietary RTG algorithm employs a further variation of this approach, using a dynamic programming edit-distance calculation for alignment of reads to a reference sequence. The alignment is *semi-global* in that it always covers the entire read but usually only covers a portion of the reference.

5.1.1 Alignment computations

Following the read mapping stage, the RTG aligner is presented with a read, a reference, a putative start position, and the frame. An alignment is produced with a corrected start position, which is subsequently converted by RTG into a SAM record.

If the corrected start position differs from the putative start position, then the alignment may be recomputed starting from the new start position (this is because slightly different alignments can result depending on the start position given to the aligner). Later stages in the RTG pipeline may decide to discard the alignment or to identify alignments together (for the purpose of removing duplicates). But the reference is always presented in the forward-sense and the edit-distance code itself makes the necessary adjustment for reverse complement cases. This avoids having to construct a reverse complement copy of the reference.

The matrix is initialized in a manner such that a small shift in start position incurs no penalty, but as the shift increases, an increasing penalty is applied. If after completing the alignment, such a path is still chosen, then the penalty is removed from the resulting score. This penalty is designed to prevent the algorithm from making extreme decisions like deleting the entire reference.

5.1.2 Alignment scoring

The basic costs used in the alignment are 0 for a match, 9 for a substitution, 19 for initiating an insertion or deletion, and 1 for continuing an insertion or deletion. All of these except for the match score can be overridden using the `--mismatch-penalty` parameter (for substitutions), the `--gap-open-penalty` parameter (for initiating an insertion or deletion) and the `--gap-extend-penalty` parameter (for continuing an insertion or deletion).

[1] Levenshtein, V. I.(1966) Binary codes capable of correcting deletions, insertions and reversal. Soviet Physics Doklady, 6:707-710.

[2] Needleman, S. B and Wunsch, C. D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:443-453

[3] Smith, T. F. and Waterman, M. S. (1981) Identification of common molecular subsequences. Journal of Molecular Biology. 147:195-197.

By default the penalty for matching an unknown nucleotide (n) in the read or reference is 5, however this can be overridden using the `--unknowns-penalty` flag. Note that regardless of the penalty for unknown nucleotides the CIGAR will always indicate unknown bases as mismatches. Occasionally, alignments may go outside the limits of the reference (that is, off the left or right ends of the reference). Such virtual bases are considered as unknown nucleotides.

Once the alignment is determined, the sum of these costs for the alignment path is reported as the *alignment score* and produced in the AS field of the corresponding SAM record. If there is a tie between an indel versus substitution operation for a particular matrix cell, then the tie is broken in favor of the substitution (except in the special case of the last column on the reference).

In the following example, the alignment score is 48, comprising a penalty of 20 for a two-nucleotide insert in the reference, 9 for a mismatch, and 19 for a one-nucleotide insert in the read. Notice there is no penalty for unknown nucleotide.

```
accg--gactctctgacgctgcncgtacgtgccaaaaataagt (reference)
||||  |||||| ||||||||||||||| ||||||||||||
accgttgactctgtgacgctgcacgtacgt-ccaaaaataagt (read)
```

In addition to the alignment score, a CIGAR is also reported in SAM records.

References for this section

5.2 Using SAM/BAM Read Groups in RTG map

It is good practice to ensure the output BAM files from the `map` command contain tracking information regarding the sample and read set. This is accomplished by specifying a read group to assign reads to. See the SAM specification for the full details of read groups, however for RTG tools, it is important to specify at least `ID`, `SM` and `PL` fields in the read group. The `ID` field should be a unique identifier for the read group, while the `SM` field should contain an identifier for the sample that was sequenced. Thus, you may have the same sample identifier present in multiple read groups (for example if the sample was sequenced in multiple lanes or by different sequencing technologies). All sample names employed by pedigree or sample-oriented commands should match the values supplied in the `SM` field, while sequencer calibration information is grouped by the read group `ID` field, and certain algorithm configuration (for example aligner penalties) may have appropriate defaults selected based on the `PL` field.

While it is possible to post-process BAM files to add this information, it is more efficient to supply the read group information either during mapping or when formatting read data to SDF. For the RTG software, the read group can either be specified in a string on the command line or by creating a file containing the SAM-formatted read group header entry to be passed to the command.

To specify a read group on the command line directly use a string encapsulated in double quotes using `\t` to denote a TAB character:

```
$ rtg map ... --sam-rg "@RG\tID:SRR002978\tSM:NA19240\tPL:ILLUMINA" ...
```

To specify a read group using a file, create or use a file containing a single SAM-formatted read group header line:

```
$ echo -e "@RG\tID:SRR002978\tSM:NA19240\tPL:ILLUMINA" > mysamrg.txt
$ cat mysamrg.txt
@RG ID:SRR002978 SM:NA19240 PL:ILLUMINA
$ rtg map ... --sam-rg mysamrg.txt ...
```

Note that when supplying read group headers in a file literal TAB characters, not `\t`, are required to separate fields.

The platform tags supported by RTG are `ILLUMINA` for Illumina reads, `COMPLETE` for Complete Genomics version 1 reads, `COMPLETEGENOMICS` for Complete Genomics version 2 reads, `LS454` for 454 Life Sciences reads and `IONTORRENT` for Ion Torrent reads.

When mapping directly from SAM/BAM input with a single read group, this will automatically be set using that read group. The read group will also be automatically set when mapping from an SDF which had the read group information stored in it during formatting.

5.3 RTG reference file format

Many RTG commands can make use of additional information about the structure of a reference genome, such as expected ploidy, sex chromosomes, location of PAR regions, etc. When appropriate, this information may be stored inside a reference genome's SDF directory in a file called `reference.txt`.

The `format` command will automatically identify several common reference genomes during formatting and will create a `reference.txt` in the resulting SDF. However, for non-human reference genomes, or less common human reference genomes, a pre-built reference configuration file may not be available, and will need to be manually provided in order to make use of RTG sex-aware pipeline features.

Several example `reference.txt` files for different human reference versions are included as part of the RTG distribution in the `scripts` subdirectory, so for common reference versions it will suffice to copy the appropriate example file into the formatted reference SDF with the name `reference.txt`, or use one of these example files as the basis for your specific reference genome.

To see how a reference text file will be interpreted by the chromosomes in an SDF for a given sex you can use the `sdfstats` command with the `--sex` flag. For example:

```
$ rtg sdfstats --sex male /data/human/ref/hg19

Location            : /data/human/ref/hg19
Parameters          : format -o /data/human/ref/hg19 -I chromosomes.txt
SDF Version         : 11
Type                : DNA
Source              : UNKNOWN
Paired arm          : UNKNOWN
SDF-ID              : b6318de1-8107-4b11-bdd9-fb8b6b34c5d0
Number of sequences : 25
Maximum length      : 249250621
Minimum length      : 16571
Sequence names      : yes
N                   : 234350281
A                   : 844868045
C                   : 585017944
G                   : 585360436
T                   : 846097277
Total residues      : 3095693983
Residue qualities   : no

Sequences for sex=MALE:
chrM POLYPLOID circular 16571
chr1 DIPLOID linear 249250621
chr2 DIPLOID linear 243199373
chr3 DIPLOID linear 198022430
chr4 DIPLOID linear 191154276
chr5 DIPLOID linear 180915260
chr6 DIPLOID linear 171115067
chr7 DIPLOID linear 159138663
chr8 DIPLOID linear 146364022
chr9 DIPLOID linear 141213431
chr10 DIPLOID linear 135534747
chr11 DIPLOID linear 135006516
chr12 DIPLOID linear 133851895
chr13 DIPLOID linear 115169878
chr14 DIPLOID linear 107349540
chr15 DIPLOID linear 102531392
```

```
chr16 DIPLOID linear 90354753
chr17 DIPLOID linear 81195210
chr18 DIPLOID linear 78077248
chr19 DIPLOID linear 59128983
chr20 DIPLOID linear 63025520
chr21 DIPLOID linear 48129895
chr22 DIPLOID linear 51304566
chrX HAPLOID linear 155270560 ~=chrY
    chrX:60001-2699520 chrY:10001-2649520
    chrX:154931044-155260560 chrY:59034050-59363566
chrY HAPLOID linear 59373566 ~=chrX
    chrX:60001-2699520 chrY:10001-2649520
    chrX:154931044-155260560 chrY:59034050-59363566
```

The reference file is primarily intended for XY sex determination but should be able to handle ZW and X0 sex determination also.

The following describes the reference file text format in more detail. The file contains lines with TAB separated fields describing the properties of the chromosomes. Comments within the `reference.txt` file are preceded by the character `#`. The first line of the file that is not a comment or blank must be the version line.

```
version1
```

The remaining lines have the following common structure:

```
<sex> <line-type>     <line-setting>...
```

The sex field is one of `male`, `female` or `either`. The line-type field is one of `def` for default sequence settings, `seq` for specific chromosomal sequence settings and `dup` for defining pseudo-autosomal regions. The *line-setting* fields are a variable number of fields based on the line type given.

The default sequence settings line can only be specified with `either` for the sex field, can only be specified once and must be specified if there are not individual chromosome settings for all chromosomes and other contigs. It is specified with the following structure:

```
either        def     <ploidy>        <shape>
```

The *ploidy* field is one of `diploid`, `haploid`, `polyploid` or `none`. The *shape* field is one of `circular` or `linear`.

The specific chromosome settings lines are similar to the default chromosome settings lines. All the sex field options can be used, however for any one chromosome you can only specify a single line for `either` or two lines for `male` and `female`. They are specified with the following structure:

```
<sex> seq     <chromosome-name>       <ploidy>        <shape> [allosome]
```

The *ploidy* and *shape* fields are the same as for the default chromosome settings line. The *chromosome-name* field is the name of the chromosome to which the line applies. The *allosome* field is optional and is used to specify the allosome pair of a haploid chromosome.

The pseudo-autosomal region settings line can be set with any of the *sex* field options and any number of the lines can be defined as necessary. It has the following format:

```
<sex> dup     <region>        <region>
```

The regions must be taken from two haploid chromosomes for a given sex, have the same length and not go past the end of the chromosome. The regions are given in the format `<chromosome-name>:<start>-<end>` where start and end are positions counting from one and the end is non-inclusive.

An example for the HG19 human reference:

```
# Reference specification for hg19, see
# http://genome.ucsc.edu/cgi-bin/hgTracks?hgsid=184117983&chromInfoPage=
```

```
version 1
# Unless otherwise specified, assume diploid linear. Well-formed
# chromosomes should be explicitly listed separately so this
# applies primarily to unplaced contigs and decoy sequences
either        def     diploid linear
# List the autosomal chromosomes explicitly. These are used to help
# determine "normal" coverage levels during mapping and variant calling
either        seq     chr1    diploid linear
either        seq     chr2    diploid linear
either        seq     chr3    diploid linear
either        seq     chr4    diploid linear
either        seq     chr5    diploid linear
either        seq     chr6    diploid linear
either        seq     chr7    diploid linear
either        seq     chr8    diploid linear
either        seq     chr9    diploid linear
either        seq     chr10   diploid linear
either        seq     chr11   diploid linear
either        seq     chr12   diploid linear
either        seq     chr13   diploid linear
either        seq     chr14   diploid linear
either        seq     chr15   diploid linear
either        seq     chr16   diploid linear
either        seq     chr17   diploid linear
either        seq     chr18   diploid linear
either        seq     chr19   diploid linear
either        seq     chr20   diploid linear
either        seq     chr21   diploid linear
either        seq     chr22   diploid linear
# Define how the male and female get the X and Y chromosomes
male  seq     chrX    haploid linear  chrY
male  seq     chrY    haploid linear  chrX
female        seq     chrX    diploid linear
female        seq     chrY    none    linear
#PAR1 pseudoautosomal region
male  dup     chrX:60001-2699520      chrY:10001-2649520
#PAR2 pseudoautosomal region
male  dup     chrX:154931044-155260560        chrY:59034050-59363566
# And the mitochondria
either        seq     chrM    polyploid       circular
```

As of the current version of the RTG software the following are the effects of various settings in the `reference.txt` file when processing a sample with the matching sex.

A ploidy setting of `none` will prevent reads from mapping to that chromosome and any variant calling from being done in that chromosome.

A ploidy setting of `diploid`, `haploid` or `polyploid` does not currently affect the output of mapping.

A ploidy setting of `diploid` will treat the chromosome as having two distinct copies during variant calling, meaning that both homozygous and heterozygous diploid genotypes may be called for the chromosome.

A ploidy setting of `haploid` will treat the chromosome as having one copy during variant calling, meaning that only haploid genotypes will be called for the chromosome.

A ploidy setting of `polyploid` will treat the chromosome as having one copy during variant calling, meaning that only haploid genotypes will be called for the chromosome. For variant calling with a pedigree, maternal inheritance is assumed for polyploid sequences.

The shape of the chromosome does not currently affect the output of mapping or variant calling.

The allosome pairs do not currently affect the output of mapping or variant calling (but are used by simulated data generation commands).

The pseudo-autosomal regions will cause the second half of the region pair to be skipped during mapping. During

variant calling the first half of the region pair will be called as diploid and the second half will not have calls made for it. For the example `reference.txt` provided earlier this means that when mapping a male the X chromosome sections of the pseudo-autosomal regions will be mapped to exclusively and for variant calling the X chromosome sections will be called as diploid while the Y chromosome sections will be skipped. There may be some edge effects up to a read length either side of a pseudo-autosomal region boundary.

5.4 RTG taxonomic reference file format

When using a metagenomic reference SDF in the `species` command, a taxonomy can be applied to impute associations between reference sequences. This is done using two files contained in the SDF directory. The first file (`taxonomy.tsv`) contains an RTG taxonomy tree and the second file (`taxonomy_lookup.tsv`) contains a mapping between taxon IDs and reference sequence names. Using a reference SDF containing these files allows the output of certain commands to include results at different taxonomic ranks allowing analysis at differing taxonomic levels.

Pre-constructed metagenomic reference SDFs in this format will be available from our website (http://www.realtimegenomics.com). For custom reference SDF creation, the `ncbi2tax` and `taxfilter` commands can assist the creation of a custom `taxonomy.tsv` file from an NCBI taxonomy dump. The `taxstats` command can check the validity of a metagenomic reference SDF.

5.4.1 RTG taxonomy file format

The RTG taxonomy file format describes the structure of the taxonomy tree. It contains multiple lines with each line being either a comment or holding data required to describe a node in the taxonomy tree.

Lines starting with a '#' are comments and do not contain data. They may appear anywhere through the file. The first line of the file must be a comment containing the RTG taxonomy file version number.

Each data line in the file represents a node in the taxonomy tree and is comprised of four tab separated values. The values on each line are:

1. The unique taxon ID of the node in the tree. This must be an integer value greater than or equal to 1.
2. The taxon ID of the parent of this node. This must be an integer value corresponding to another node in the tree.
3. The rank of the node in the taxonomy. This is a free format string that can contain any character other than a tab.
4. The name of the node in the taxonomy. This is a free format string that can contain any character other than a tab.

The root of the tree is special and must have a taxon ID of 1. Since the root has no parent it can have a parent ID of either 1 (itself) or -1. The RTG taxonomy file should contain a complete and fully connected tree that has a single root and no loops.

An example of the first few lines of a `taxonomy.tsv` file:

```
#RTG taxonomy version 1.0
#taxID        parID   rankname
1-1   no rank root
129081        no rank unclassified sequences
40816912908   no rank metagenomes
410657408169  no rank ecological metagenomes
527640410657  species microbial mat metagenome
1315671       no rank cellular organisms
2131567       superkingdom    Bacteria
11172 phylum  Cyanobacteria
```

5.4.2 RTG taxonomy lookup file format

The RTG taxonomy lookup file format associates SDF sequence names with taxon IDs in the taxonomy tree. It contains one line for every sequence in the SDF, with each line containing two tab separated values.

The values on each line are:

1. The taxonomy node ID that the sequence is associated with. This must be an integer value that corresponds to a node ID from the taxonomy tree.
2. The name of the sequence as it appears in the SDF. (These can be discovered using the `--lengths` option of the `sdfstats` command)

A single taxon ID may be associated with multiple sequence names. This is a way to group the chromosomes and plasmids belonging to a single organism.

An example of some lines from a `taxonomy_lookup.tsv` file:

```
1219061gi|407098174|gb|AMQV00000000.1|AMQV01000000
1219061gi|407098172|gb|AMQV01000002.1|
1219061gi|407098170|gb|AMQV01000004.1|
1219061gi|407098168|gb|AMQV01000006.1|
```

5.5 Pedigree PED input file format

The PED file format is a white space (tab or space) delimited ASCII file. Lines starting with # are ignored. It has exactly six required columns in the following order.

Column	Definition
Family ID	Alphanumeric ID of a family group. This field is ignored by RTG commands.
Individual ID	Alphanumeric ID of an individual. This corresponds to the Sample ID specified in the read group of the individual (`SM` field).
Paternal ID	Alphanumeric ID of the paternal parent for the individual. This corresponds to the Sample ID specified in the read group of the paternal parent (`SM` field).
Maternal ID	Alphanumeric ID of the maternal parent for the individual. This corresponds to the Sample ID specified in the read group of the maternal parent (`SM` field).
Sex	The sex of the individual specified as using 1 for male, 2 for female and any other number as unknown.
Phenotype	The phenotype of the individual specified using -9 or 0 for unknown, 1 for unaffected and 2 for affected.

Note: The PED format is based on the PED format defined by the PLINK project: http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml#ped

The value '0' can be used as a missing value for Family ID, Paternal ID and Maternal ID.

The following is an example of what a PED file may look like.

```
# PED format pedigree
# fam-id ind-id pat-id mat-id sex phen
FAM01 NA19238 0 0 2 0
FAM01 NA19239 0 0 1 0
FAM01 NA19240 NA19239 NA19238 2 0
0 NA12878 0 0 2 0
```

When specifying a pedigree for the `lineage` command, use either the pat-id or mat-id as appropriate to the gender of the sample cell lineage. The following is an example of what a cell lineage PED file may look like.

```
# PED format pedigree
# fam-id ind-id pat-id mat-id sex phen
LIN BASE 0 0 2 0
LIN GENA 0 BASE 2 0
LIN GENB 0 BASE 2 0
LIN GENA-A 0 GENA 2 0
```

RTG includes commands such as `pedfilter` and `pedstats` for simple viewing, filtering and conversion of pedigree files.

5.6 RTG commands using indexed input files

Several RTG commands require coordinate indexed input files to operate and several more require them when the `--region` or `--bed-regions` parameter is used. The index files used are standard tabix or BAM index files.

The RTG commands which produce the inputs used by these commands will by default produce them with appropriate index files. To produce indexes for files from third party sources or RTG command output where the `--no-index` or `--no-gzip` parameters were set, use the RTG `bgzip` and `index` commands.

5.7 RTG output results file descriptions

RTG software produces output results in standard formats that may contain additional information for the unique requirements of a particular data analysis function.

Several of the RTG commands that output results to a directory also output a simple summary report of the results of the command in HTML format. The report file for these commands will be called `index.html` and will be contained in the output directory.

5.7.1 SAM/BAM file extensions (RTG map command output)

The Sequence Alignment/Map (SAM/BAM) format is a well-known standard for listing read alignments against reference sequences. SAM records list the mapping and alignment data for each read, ordered by chromosome (or other DNA reference sequence) location.

Note: For a thorough description of the SAM format please refer to the specification at https://samtools.github.io/hts-specs/SAMv1.pdf

---

The `map` command reports alignments in the SAM/BAM format with some minor differences.

A sample RTG SAM file is shown below, describing the relationship between a read and a reference sequence, including gaps and mismatches as determined by the RTG `map` aligner.

```
@HD VN:1.0 SO:coordinate
@PG ID:RTG VN:v2.0-EAP2.1 build 25581 (2010-03-11) CL:map -t human_REF_SDF -i␣
↪human_READS_SDF -o humanMAPPING8 -w 22 -a 2 -b 2 -c 2
@SQ SN:chr1 LN:643292
@SQ SN:chr2 LN:947102
@SQ SN:chr3 LN:1060087
@SQ SN:chr4 LN:1204112
@SQ SN:chr5 LN:1343552
@SQ SN:chr6 LN:1418244
@SQ SN:chr7 LN:1501719
@SQ SN:chr8 LN:1419563
@SQ SN:chr9 LN:1541723
@SQ SN:chr10 LN:1694445
```

```
@SQ SN:chr11 LN:2035250
@SQ SN:chr12 LN:2271477
@SQ SN:chr13 LN:2895605
@SQ SN:chr14 LN:3291006
1035263       0       chr1    606     255     2X18=1X11=1X15= *       0       0
↪   AATACTTTTCATTCTTTACTATTACTTACTTATTCTTACTTACTTACT        *       AS:i:4
↪NM:i:4  IH:i:1  NH:i:1
1572864       16      chr1    2041    255     48=     *       0       0
↪TACTTACTTTCTTCTTACTTATGTGGTAATAAGCTACTCGGTTGGGCA        *       AS:i:0  NM:i:0
↪IH:i:1  NH:i:1
2649455       0       chr1    3421    255     2X46=   *       0       0
↪AAGTACTTCTTAGTTCAATTACTATCATCATCTTACCTAATTACTACT        *       AS:i:2  NM:i:2
↪IH:i:1  NH:i:1
```

RTG identifies each query name (or QNAME) with an RTG identifier, which replaced the original identifier associated with the read data. The RTG `samrename` utility is used to relabel the alignment records with the original sequence names.

The CIGAR format has evolved from the original SAM specification. Formerly, a CIGAR string might appear as `283M1D1I32N8M`, where `M` indicated a match or mismatch, `I` indicated an insertion, `D` indicated a deletion, and `N` indicated a skipped region.

In the sample SAM file above, the RTG CIGAR score characters are modified as represented by the string `2X18=1X11=1X15=`, where `X` indicates a mismatch, = indicates a match, `I` indicates an insertion, and `D` indicates a deletion. Obviously, this provides more specificity around the precise location of the mismatch in the alignment.

Notice that optional fields are reported as in SAM for alignment score (`AS`), number of nucleotide differences (`NM`), number of reported alignments for a particular read (`NH`), number of stored alignments (`IH`) and depending on flag settings, the field containing the string describing mismatching positions (`MD`) may be included. The alignment score is calculated and reported for RTG as described in *RTG gapped alignment technical description*.

The following list describes RTG SAM/BAM file characteristics that may depart from or be undescribed in the SAM specification.

- Paired-end sequencing data
- `FLAG` is set to 0x02 in properly paired reads and unset for unmated or unmapped reads.
- For all non-uniquely mapped reads `FLAG` 0x100 is set.
- Unmated and unmapped reads will have the `FLAG` 0x08 set to reflect whether the mate has been mapped, however `RNEXT` and `PNEXT` will always be "*".
- For mapped reads, the SAM standard `NH` attribute is used, even for uniquely mapped reads (`NH:i:1`).
- Single-end sequencing data
- For all non-uniquely mapped reads `FLAG` 0x100 is set.
- For mapped reads, the SAM standard `NH` attribute is used, even for uniquely mapped reads (`NH:i:1`).
- Unmapped reads
- `RNAME` and `CIGAR` are set as "*".
- `POS`, and `MAPQ` are set as 0.

For mated records, the `XA` attribute contains the sum of the alignment scores for each arm. It is this score that is used to determine the equality for the max top results for mated records. All the mated records for a read should have the same `XA` score.

In addition, for unmapped read arms, the optional attribute `XC` may be displayed in SAM/BAM files, using a character code that indicates the internal read mapping stage where the read arm was discarded as unmated or unmapped. If this is reported, it means that the read arm had matching hits at one point during the mapping phase.

Single-end SAM character codes include:

Character	Definition
XC:A:B	Indicates that the number of raw index hits for the read exceeded the internal threshold of 65536.
XC:A:C	Indicates that after initial ranking of hits for the read, too many hits were present (affected by --max-top-results).
XC:A:D	Indicates that after alignment scores are calculated, the $\leq N$ remaining hits were discarded because they exceeded the mismatches threshold (affected by --max-mismatches).
XC:A:E	Indicates that there were good scoring hits, but the arm was discarded because there were too many of these hits (affected by --max-top-results).

Paired-end SAM character codes include:

Character	Definition
XC:A:B	Indicates that the number of raw index hits for the read exceeded the internal threshold of 65536.
XC:A:C	Indicates that there were index matches for the read arm, but no potential mated hits were found (affected by --min-fragment-size and --max-fragment-size), and after ranking candidate unpaired results there were too many hits (affected by --max-top-results).
XC:A:d	Indicates that potential mated hits were found for this read arm with its mate, but were discarded because they exceeded the mismatches threshold (affected by --max-mated-mismatches).
XC:A:D	Indicates that no potential mated hits were found, and after alignment scores are calculated, the ($\leq N$) remaining hits were discarded because they exceeded the mismatches threshold (affected by --max-unmated-mismatches).
XC:A:e	Indicates that good scoring hits were found for this read arm with its mate, but were discarded because there were too many hits at the best score (affected by --max-top-results).
XC:A:E	Indicates that no potential mated hits were found, there were good scoring unmated hits, but the arm was discarded because there were too many of these hits (affected by --max-top-results).

5.7.2 SAM/BAM file extensions (RTG cgmap command output)

In addition to the file extensions described for the map command in *SAM/BAM file extensions (RTG map command output)*, the cgmap command also outputs several additional fields specific to the nature of Complete Genomics reads.

A sample RTG SAM file is shown below, describing the relationship between some reads and a reference sequence, including gaps, mismatches and overlaps as determined by the RTG cgmap aligner.

```
@HD VN:1.0 SO:coordinate
@PG ID:RTG PN:RTG VN:v2.3 CL:cgmap -i bac_READS_SDF -t bac_REF_SDF -o bac_MAPPING -
↪e 7 -E 5
@SQ SN:bac LN:100262
1 179 bac 441 55 24=5N10= = 765 324 TGACGCCTCTGCTCTTGCAAGTCNTTCACATTCA 544400/31/
↪1\*2\*858154468!073./66222 AS:i:0 MQ:i:255 XU:Z:5=1B19=1R5N10= XQ:Z:+ XA:i:0␣
↪IH:i:1 NH:i:1
1 115 bac 765 55 10=5N8=1I14= = 441 -324 ANAGAACTGGAACCATTCATGGAAGGCCAGTGA 5!5/1,+!
↪431/..,153002076-13435001 AS:i:0 MQ:i:255 XU:Z:1=1R5=1R2=5N8=1I11=2B5= XQ:Z:74␣
↪XR:Z:TA XA:i:0 IH:i:1 NH:i:1
83 179 bac 4963 55 3=1X19=5N10= = 5257 294 GGAAGGAGTGCTGCAGGCCGACCCTCATGGAGA 42062-
↪51/4-1,55.010456-27/2711032 AS:i:1 MQ:i:255 XU:Z:3=1X1=2B20=5N10= XQ:Z:-. XR:Z:A␣
↪XA:i:1 IH:i:1 NH:i:1
83 115 bac 5257 55 10=5N25= = 4963 -294 CCTCCTAGCGGTACATCTCCAGCCCCTTCCTAGNA␣
↪55541\*,-/3+1,2,13525167".21806010!2 AS:i:0 MQ:i:255 XU:Z:10=5N23=1R1= XA:i:1␣
↪IH:i:1 NH:i:1
```

The XU field is an extended CIGAR format which has additional characters for encoding extra information about a Complete Genomics read. The extra characters are B for encoding a backstep on the reference (overlap in the read), T for an unknown nucleotide in the reference and R for an unknown nucleotide in the read. In the case where both the reference and the read have an unknown nucleotide at the same place the R character is used.

The XQ field is the quality in the same format as the SAM QUAL field for the nucleotides in the overlap region of the read. It is present when backsteps exist in the extended CIGAR field.

The XR field contains the nucleotides from the read which differ from the reference (read delta). It is present when there are mismatches, inserts, unknowns on the reference or soft clipping represented in the extended CIGAR.

Using these three additional fields with the QUAL field, position that the read mapped to, and the reference, it is possible to reconstruct the original Complete Genomics read.

For example:

```
Record:
Position 4963 42062-51/4-1,55.010456-27/2711032 XU:Z:3=1X1=2B20=5N10= XQ:Z:-.␣
↪XR:Z:A

Reference 4960-5010:
CCTGGAGG  GAGTGCTGCAGGCCGACCAGCAACTCATGGAGAAGACCAAGG
   GGAAGGGGAGTGCTGCAGGCCGACC     CTCATGGAGA
   42062-.-51/4-1,55.010456-_____27/2711032

Flattened Read from SAM file:
GGAAGGAGTGCTGCAGGCCGACCCTCATGGAGA
42062-51/4-1,55.010456-27/2711032

Reconstructed Read:
GGAAGGGGAGTGCTGCAGGCCGACCCTCATGGAGA
42062-.-51/4-1,55.010456-27/2711032
```

This example shows how the mismatch and read delta replace the nucleotide from the reference to form the read. It also shows that the backstep in the extended CIGAR is used to record overlap in the read. Note that the number of additional quality characters in the corresponds to the number of backsteps and that the additional quality characters will always be inserted into the read qualities on the inner-most side of the overlap region.

```
Record:
Position 65 5!5/1,+!431/..,153002076-134735001 XU:Z:1=1R5=1T2=5N8=1I11=2B5=␣
↪XR:Z:TA XQ:Z:74

Reference 760-810
AGAGGAGAGAACNGGTTTGGAACCATTC TGGAAGGCCAG  TGAGCTGTGTT
     ANAGAACTGG_____AACCATTCATGGAAGGCCAGAGTGA
     5!5/1,+143_____1/..,153002076-1347435001

Flattened Read from SAM file:
ANAGAACTGGAACCATTCATGGAAGGCCAGTGA
5!5/1,+!431/..,153002076-13435001

Reconstructed Read:
ANAGAACNGGAACCATTCATGGAAGGCCAGAGTGA
5!5/1,+1431/..,153002076-1347435001
```

This example shows how the R character in the extended CIGAR corresponds to an unknown in the read and how the T character corresponds to an unknown in the reference. Note that when there is an unknown in the reference but not in the read the nucleotide is included in the read delta as are inserted nucleotides. Also note that although the backstep is used in this case to reconstruct part of the outside five nucleotides the overlap quality characters still correspond to the inside nucleotides.

5.7.3 Small-variant VCF output file description

The snp, family, somatic and population commands call single nucleotide polymorphisms (SNPs), multiple nucleotide polymorphisms (MNPs), and indels for a single individual, a family of individuals or a cancer and normal pair respectively. At each position in the reference, a base pair determination is made based on statisti-

cal analysis of the accumulated read alignments. The predictions and accompanying statistics are reported in a text-based output file named `snps.vcf`.

Note: RTG variant calls are stored in VCF format (version 4.1). For more information about the VCF format, refer to the specification online at: https://samtools.github.io/hts-specs/VCFv4.1.pdf

The `snps.vcf` output file displays each variant called with confidence. The location and type of the call, the base pairs (reference and called), and a confidence score are standard output in the `snps.vcf` output file. Additional support statistics in the output describe read alignment evidence that can be used to evaluate confidence in the called variants.

The commands also produce a `summary.txt` file which has simple counts of the variants detected and some ratios which can be used as a quick indication of SNP calling performance.

The following sample `snps.vcf` file is an example of the output produced by an RTG SNP call run. Each line in a `snps.vcf` output has tab-separated fields and represents a SNP variation calculated from the mapped reads against the reference genome.

This file represents the variations per chromosome as a result of the SAM/BAM mapped alignments against a reference genome.

```
##fileformat=VCFv4.1
##fileDate=20110524
##source=RTGv2.2 build 35188 (2011-05-18)
##CL=snp -o snp-hslo-18-u -t hst1 hslo-18-u/alignments.bam
##RUN-ID=b1f96b37-7f77-4d74-b472-2a36ba21397e
##reference=hst1
##contig=<ID="chr1",length=207900>
##INFO=<ID=XRX,Number=0,Type=Flag,Description="RTG variant was called using␣
↪complex caller">
##INFO=<ID=CT,Number=1,Type=Integer,Description="Coverage threshold that was␣
↪applied">
##FILTER=<ID=OC,Description="Coverage threshold exceeded">
##FILTER=<ID=RC,Description="RTG variant is a complex region">
##FILTER=<ID=RX,Description="RTG variant contained within hypercomplex region">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=RE,Number=1,Type=Float,Description="RTG Total Error">
##FORMAT=<ID=AR,Number=1,Type=Float,Description="Ambiguity Ratio">
##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype Quality">
##FORMAT=<ID=RS,Number=.,Type=String,Description="RTG Support Statistics">
#CHROM    POS     ID    REF      ALT         QUAL   FILTER  INFO    FORMAT    SAMPLE
chr1      43230   .     A        G           17.1   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:7:0.280:0.000:17.1:G,7,0.280
chr1      43494   .     TTTAAAT  TT:CTTAAAC  181.6  PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/2:10:0.428:0.024:15.3:CTTAAC,5,0.373
chr1      43638   .     T        C           15.3   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:6:0.210:0.000:15.3:C,6,0.210
chr1      43918   .     C        T           11.4   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:5:0.494:0.000:11.4:T,5,0.494
chr1      44038   .     C        T           18.9   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:8:0.672:0.000:18.9:T,8,0.672
chr1      44173   .     A        G           12.0   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:5:0.185:0.000:12.0:G,5,0.185
chr1      44218   .     TCCTCCA  ACCACCT     385.4  PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:43:3.925:0.015:80.8:ACCACCT,5,0.003,CCCCCCA,1,0.631,
↪CCCTCCA,1,0.631,   TCCTCCA,30,2.024,TCCTTCA,4,0.635,~A,1,0.000,~TCCA,1,0.001
chr1      44329   .     A        G           12.2   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:5:0.208:0.000:12.2:G,5,0.208
chr1      44502   .     T        C           6.0    PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:3:0.160:0.000:6.0:C,3,0.160
chr1      44533   .     G        A           13.7   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   1/1:6:0.421:0.000:13.7:A,6,0.421
```

```
chr1      202801 .     A        G           15.8   PASS    .       ␣
↪GT:DP:RE:AR:GQ:RS   0/1:66:30.544:0.000:15.8:A,50,24.508,C,1,0.502,G,13,4.529,T,
↪2,1.004
```

Note: The VCF specification defines the semantics of the `QUAL` column differently for records that contain any `ALT` alleles from those which do not, and that the `QUAL` column is also defined as a score applying across the set of all samples in the VCF. Thus for multi-sample calling commands such as `family`, `population`, `somatic`, etc, the `QUAL` score is not necessarily an indication of the of quality the call of any particular sample.

RTG adds custom fields to the VCF format to better account for some of its unique variant calling features, described in the tables below. The exact set of fields used depend on the module run, with some fields only appropriate and present for family calling, somatic calling, etc.

Table : RTG VCF file FILTER fields

Value	Description
`PASS`	Standard VCF `PASS`, if variant meets all the filtering criteria.
`OC`	A predicted variation that has exceeded the maximum coverage filter threshold.
`a<number>`	A predicted variation that had greater than the given percentage of ambiguously mapped reads overlapping it. The number in the value is the percentage specified by the `--max-ambiguity` flag.
`RCEQUIV`	A predicted variation that is the same as a previous variant within a homopolymer or repeat region.
`RC`	The variant caller encountered a complex looking situation. A typical example would be a long insert. Some complex regions may result in simple calls.
`RX`	This call was made within a long complex region. Note that no attempt is made to generate complex calls in very long complex regions.
`IONT`	A predicted variation that failed to pass homopolymer constraints specific to IonTorrent reads.
`OTHER`	The variant was filtered for an unknown reason.
`AVR<number>`	A predicted variation that had less than the given value for the AVR score. The number in the value is the minimum AVR score specified by the `--min-avr-score` flag.
`BED`	The predicted variant falls outside the BED calling regions defined by the `--filter-bed` flag.

Table : RTG VCF file INFO fields

Value	Description
`NCS=<value>`	The Phred scaled posterior probability that the variant at this site is present in the cancer, output by the `somatic` command.
`LOH=<value>`	The value shows on a scale from -1 to 1 if the evidence of a call would suggest a loss of heterozygosity. A long run of high values is a strong indicator of a loss of heterozygosity event.
`DP=<depth>`	The combined read depth of multi-sample variant calls.
`DPR=<ratio>`	The ratio of combined read depth to the expected combined read depth.
`XRX`	Indicates the variant was called using the RTG complex caller. This means that a realignment of the reads relative to the reference and each other was required to make this call.
`RCE`	Indicates the variant is the same as one or more other variants within a homopolymer or repeat region.
`NREF`	Indicates the variant is called at a site where the reference is unknown, and so some other scores that require exact knowledge of the reference may not be produced.
`CT=<value>`	The maximum coverage threshold that was applied when the given variant has been filtered for being over the coverage threshold.
`AC`	The standard VCF allele count in genotypes field. For each ALT allele, in the same order as listed, the count of the allele in the genotypes.
`AN`	The standard VCF total number of alleles in called genotypes field.
`STRL`	The number of adjacent simple tandem repeats on the reference sequence.
`STRU`	The length of the repeating unit in a simple tandem repeat.

Table : RTG VCF file FORMAT fields

Value	Description
GT	The standard VCF format genotype field.
DP	The standard VCF format read depth field.
DPR	The ratio of read depth to the expected read depth.
VA	The allele index (using same numbering as the GT field) of the most frequent non-REF allele. This allele may not necessarily be part of the genotype that was actually called.
RE	The total error across bases of the reads at the SNP location. This is a corrective factor calculated from the r and q read mapping quality scores that adjusts the level of confidence higher or lower relative to read depth.
AR	The ratio of reads contributing to the variant that are considered to be ambiguous to uniquely mapped reads.
RQ	The Phred scaled posterior probability that the sample is not identical to the reference.
GQ	The standard VCF format genotype quality field. This is the Phred scaled posterior score of the call. It is not necessarily the same as the QUAL column score.
GQD	The genotype quality divided by the read depth of the sample.
QD	The quality field divided by the sum of the read depth for all samples.
DN	Indicates with a value of Y if the call for this sample is a putative de novo mutation, or N to indicate that the sample is not a de novo mutation. Note that even in cases when the GT of the sample and the parents would otherwise seem to indicate a de novo mutation, this field may be set to N when the variant caller has assigned a sufficiently low score to the likelihood that a de novo event has occurred.
DNP	The Phred scaled probability that the call for this sample is due to a de novo mutation.
OCOC	The count of evidence that is considered contrary to the call made for this sample, observed in the original sample. For example, in a normal-cancer somatic call of 0/0 -> 1/0, the OCOC value is the count of the somatic (1) allele in the normal sample. Usually a high OCOC value indicates an unreliable call.
OCOF	The fraction of evidence that is considered contrary to the call made for this sample, observed in the original sample. For example, in a somatic call of 0/0 -> 1/0, the OCOF value is the fraction of the somatic (1) allele in the normal sample. The OCOF and OCOC attributes are also applicable to de novo calls, where the evidence in the parents for the de novo allele is considered contrary. Usually a high OCOF value indicates an unreliable call.
DCOC	The count of evidence that is considered contrary to the call made for this sample, observed in the derived sample. For example, in a normal-cancer somatic call of 0/1 -> 2/0, the DCOC value is the count of the germline (1) allele in the somatic sample. In cases of high sample purity, a high DCOC value may indicate an unreliable call.
DCOF	The fraction of evidence that is considered contrary to the call made for this sample, observed in the derived sample. For example, in a somatic call of 0/1 -> 2/0, the DCOF value is the fraction of the germline (1) allele in the somatic sample. The DCOF and DCOC attributes are also applicable to pedigree aware calls, where the evidence of non-inherited parental alleles in the child is considered contrary. In cases of high sample purity, a high DCOF value may indicate an unreliable call.
ABP	The Phred scaled probability that allele imbalance is present in the call.
SBP	The Phred scaled probability that strand bias is present in the call.
RPB	The Phred scaled probability that read position bias is present in the call.
PPB	The Phred scaled probability that bias in the proportion of alignments that are properly paired is present in the call.
PUR	The ratio of placed unmapped reads to mapped reads.

Continued on next page

Table 5.1 – continued from previous page

Value	Description
RS	Statistical information about the evidence for the prediction which consists of a variable number of groups of three fields, each separated by commas. The three fields are allele, count, and the sum of probability error (computed from the Phred quality). The sum of counts should equal DP and the sum of the errors should equal RE.
DH	An alternative disagreeing hypothesis in the same format as the genotype field. This can occur when a sample intersects multiple families in a pedigree when doing population calling.
AD	The allelic depths for the reference and alternate alleles in the order listed.
ADE	The allelic depths for the reference and alternate alleles in the order listed, after adjusting for poor base quality and mapping quality.
AQ	The sum of the quality of evidence (including base quality and mapping quality) for the reference and alternate alleles in the order listed.
MEANQAD	The difference in the mean AQ between the two called alleles.
SSC	The score for the somatic mutation specified by the GT field.
SS	The somatic status of the genotype for this sample. A value of 0 indicates none or wild type, a value of 1 indicates a germline variant, and a value of 2 indicates a somatic variant.
GL	The log10 scaled likelihoods for all possible genotypes given the set of alleles defined in the REF and ALT fields as defined in the VCF specifications.
VAF	The VAF field contains the estimated variant allelic fraction of each alternate allele, in the order listed.
VAF1	The VAF1 field contains the estimated variant allelic fraction of the most abundant non-reference allele. This attribute may be more suitable for AVR model building and filtering than the multi-valued VAF annotation.
IC	The inbreeding coefficient for the site.
EP	The Phred scaled probability that the site is not in Hardy-Weinberg equilibrium.
LAL	The length of the longest allele for the site.
NAA	The number of alternate alleles for the site.
PD	The ploidy of the sample.
ZY	The zygosity of the sample.
RA	Categorizes the call as hom-ref (RR), hom-alt (AA), het-ref (RA), or het-alt (AB), independent of phase or ALT allele indices.
QA	Sum of quality of the alternate observations.
CLUS	The number of variants in this sample within five bases of the current variant.
AVR	The adaptive variant rescoring value. It is a value between 0 and 1 that represents the probability that the variant for the sample is correct.

The following examples of what a variant call may look like only includes the bare minimum sample field information for clarity.

```
#Examples of calls:
g1  64  .  T    A        74.0  PASS    .       GT  1/1  #Homozygous SNP
g1  9   .  GAC  G        9.0   PASS    .       GT  1/1  #Homozygous deletion
g1  16  .  A    ACGT     27.0  PASS    .       GT  1/1  #Homozygous insertion
g1  54  .  T    TA       14.0  PASS    .       GT  1/0  #Heterozygous insertion
g1  17  .  ACGT A        71.0  PASS    .       GT  0/1  #Heterozygous deletion
g1  61  .  TTA  GCG,AAT  74.0  PASS    .       GT  1/2  #Heterozygous MNP
g1  3   .  A    .        11.0  PASS    .       GT  0/0  #Equality call
g1  32  .  CGT  .        89.0  PASS    .       GT  0/0  #Equality call
g1  88  .  A    G        249.5 PASS    .       GT  1    #Haploid SNP
g1  33  .  A    T        20.0  PASS    RCE     GT  1/1  #Variant which is␣
↪equivalent to other variants
g1  42  .  A    T        13.0  RCEQUIV RCE     GT  1/1  #Variant filtered due to␣
↪equivalence to other variants
g1  45  .  A    C        5.0   OC      CT=100  GT  1/1  #Variant which exceeds the␣
↪coverage threshold of 100
g1  76  .  G    C        10.0  a10.0   .       GT  1/1  #Variant which had␣
↪ambiguous mappings overlapping it
```

```
g1  74  .  A     .        3.0   RC      .      GT  0/0  #Complex call with no␣
↪prediction
g1  90  .  G     C        15.0  RX      .      GT  1/1  #Homozygous SNP called␣
↪within a large complex region
g1  99  .  C     G        15.0  IONT    .      GT  1/1  #Variant that failed␣
↪IonTorrent homopolymer constraints
g1  33  .  A     G,C      13.0  PASS    .      GT  0/1  0/2  0/2  1/2  #Mendelian␣
↪family call
g1  90  .  G     A        20.0  PASS    LOH=1  GT:SSC:SS  0/1  0/0:24.0:2
↪#Somatic mutation
```

In the outputs from the `family`, `population` and `somatic` commands some additional information about the samples are provided through the `PEDIGREE` and `SAMPLE` header lines.

The `family` or `population` command output includes additional sample sex and pedigree information within the headers like the following:

```
##PEDIGREE=<Child=SM_SON,Mother=SM_MOTHER,Father=SM_FATHER>
##SAMPLE=<ID=SM_SON,Sex=MALE>
##SAMPLE=<ID=SM_MOTHER,Sex=FEMALE>
##SAMPLE=<ID=SM_FATHER,Sex=MALE>
```

The pedigree information contained within VCF header fields is also used by the `mendelian`, `pedfilter`, and `pedstats` commands.

The `somatic` command output includes information about sample relationships and genome mixtures using headers like the following:

```
##PEDIGREE=<Derived=SM_TUMOR,Original=SM_BASE>
##SAMPLE=<ID=SM_BASE,Genomes=SM_BASE,Mixture=1.0,Sex=MALE,Description="Original␣
↪genome">
##SAMPLE=<ID=SM_TUMOR,Genomes=SM_BASE;SM_TUMOR,Mixture=0.2;0.8,Sex=MALE,
↪Description="Original genome;Derived genome">
```

5.7.4 Regions BED output file description

The `snp`, `family`, `population` and `somatic` commands all output a BED file containing regions that were considered to be complex.

The following sample `regions.bed` file is an example of the output produced by an RTG variant calling run. Each line in a `regions.bed` output has tab-separated fields and represents a region in the reference genome that was considered to be complex.

```
CFTR.3.70s  7 7 complex-called
CFTR.3.70s  15  15  complex-called
CFTR.3.70s  18  18  complex-called
CFTR.3.70s  21  21  complex-called
CFTR.3.70s  26  30  complex-called
CFTR.3.70s  42  45  complex-over-coverage
CFTR.3.70s  84  93  extreme-coverage
CFTR.3.70s  133 165 hyper-complex
CFTR.3.70s  169 186 complex-called
CFTR.3.70s  189 195 complex-called
CFTR.3.70s  198 198 complex-no-variant
CFTR.3.70s  300 310 complex-no-hypotheses
CFTR.3.70s  435 439 complex-too-many-hypotheses
```

The columns in order are:

1. Sequence name
2. Region start, counting from 0

3. Region end, counting from 0, not inclusive
4. Name of the region type

Table : Region type names

Value	Description
`complex-called`	Complex regions that were called using complex calling.
`hyper-complex`	Long complex regions for which no call attempt was made.
`extreme-coverage`	No calls made in this region due to extreme coverage.
`complex-over-coverage`	Complex region has greater than the maximum coverage allowed.
`complex-no-hypotheses`	No hypotheses could be created for the complex region.
`complex-no-variant`	Complex region evaluation resulted in no variants.
`complex-too-many-hypotheses`	Complex region had too many hypotheses for evaluation.

5.7.5 SV command output file descriptions

The `sv` command is used to predict the likelihood of various structural variant categories. The outputs produced are `sv_interesting.bed.gz` which is a BED format file that identifies regions that could indicate a structural variant and `sv_bayesian.tsv.gz` a tab separated format file containing prediction strengths of event models.

The following is an example of the `sv_interesting.bed.gz` file output.

```
#chr    start   end areas   maxscore    average
chr1    10    100 1   584.5470    315.8478
chr1    49760   53270 5   630.3273    380.2483
```

Table : SV_INTERESTING.BED file output column descriptions

Column	Description
`chr`	The chromosome name.
`start`	The start position in the reference chromosome.
`end`	The end position in the reference chromosome.
`areas`	The number of distinct model areas contained in the region.
`maxscore`	The maximum score reached by a model in the given region.
`average`	The average score for the model areas covered by this region.

The following is an example of the `sv_bayesian.tsv.gz` file output.

```
#Version v2.3.2 build 5c2ee18 (2011-10-05), SV bayesian output v0.1
#CL  sv --step 100 --fine-step 10 --readgroup-stats map/rgstats.tsv --template /
↪data/human/hg18.sdf -o sv map/alignments.bam
#RUN-ID  8acc8413-0daf-455d-bf9f-41d195dec4cd
#template-name    position  normal  duplicate  delete  delete-left  delete-right ␣
↪duplicate-left  duplicate-right breakpoint  novel-insertion  max-index
chr1  11  -584.5470  -1582.1325  -3538.0052  -4168.1398  584.5470  -932.2432  -
↪1219.9770  -630.1436  -664.3196  4
chr1  21  -521.2708  -1508.9226  -3617.4369  -4247.5716  521.2708  -865.7595  -
↪1156.7007  -630.1450  -671.1980  4
chr1  31  -443.5759  -1425.2073  -3626.2318  -4256.3664  443.5759  -788.1160  -
↪1079.0058  -630.1468  -662.5068  4
chr1  41  -372.8984  -1346.1013  -3676.6399  -4306.7745  372.8984  -715.6147  -
↪1008.3284  -630.1490  -662.4542  4
chr1  51  -326.3469  -1288.4120  -3790.0943  -4420.2290  326.3469  -663.2324  -961.
↪7768  -630.1516  -660.6529  4
chr1  61  -269.1376  -1223.0752  -3849.6462  -4479.7808  269.1376  -604.2883  -904.
↪5676  -630.1545  -669.2223  4
chr1  71  -201.0995  -1146.3076  -3907.0182  -4537.1529  201.0995  -534.3735  -836.
↪5295  -630.1578  -673.4514  4
chr1  81  -109.1116  -1046.1921  -3931.7913  -4561.9260  109.1116  -442.1511  -744.
↪5415  -630.1612  -669.0146  4
```

```
chr1  91  -14.6429  -941.7897  -3980.0307  -4610.1653  14.6429  -345.7608  -650.
↪0728  -630.1648  -664.5593  4
chr1  101  60.8820  -918.1163  -4095.1224  -4725.2570  -60.8820  -329.0012  -635.
↪4300  -691.0506  -719.2296  0
chr1  111  117.6873  -911.1928  -4194.5855  -4824.7202  -117.6873  -327.5866  -635.
↪4300  -747.8596  -775.8622  0
chr1  121  199.2098  -898.2490  -4380.5384  -5010.6731  -199.2098  -321.7577  -635.
↪4300  -829.3856  -857.1902  0
```

Table : SV_BAYESIAN.TSV file output column descriptions

Column	Description
template-name	The chromosome name.
position	The position in the reference chromosome.
normal	The prediction strength for the normal model.
duplicate	The prediction strength for the duplicate model.
delete	The prediction strength for the delete model.
delete-left	The prediction strength for the delete-left model.
delete-right	The prediction strength for the delete-right model.
duplicate-left	The prediction strength for the duplicate-left model.
duplicate-right	The prediction strength for the duplicate-right model.
breakpoint	The prediction strength for the breakpoint model.
novel-insertion	The prediction strength for the novel-insertion model.
max-index	The index of the model that has the maximum prediction strength for this line. The index starts from 0 meaning normal and is in the same order as the model columns.

When the `--simple-signals` parameter is set an additional file called `sv_simple.tsv.gz` is output which is a tab separated format file containing the raw signals used by the `sv` command. The following is an example of the `sv_simple.tsv.gz` output file.

```
#Version v2.3.2 build 5c2ee18 (2011-10-05), SV simple output v0.1
#CL  sv --step 100 --fine-step 10 --readgroup-stats map/rgstats.tsv --template /
↪data/human/hg18.sdf -o sv map/alignments.bam --simple-signals
#RUN-ID  8acc8413-0daf-455d-bf9f-41d195dec4cd
#template-name  position  proper-left  discordant-left  unmated-left  proper-right␣
↪ discordant-right unmated-right  not-paired  unique  ambiguous  n-count
chr1  11  17.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  18.0000  0.
↪0000  0.0000
chr1  21  14.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  15.0000  0.
↪0000  0.0000
chr1  31  13.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  13.0000  0.
↪0000  0.0000
chr1  41  10.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  11.0000  0.
↪0000  0.0000
chr1  51  11.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  11.0000  0.
↪0000  0.0000
chr1  61  12.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  12.0000  0.
↪0000  0.0000
chr1  71  11.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  11.0000  0.
↪0000  0.0000
chr1  81  17.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  17.0000  0.
↪0000  0.0000
chr1  91  17.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  17.0000  0.
↪0000  0.0000
chr1  101  9.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  10.0000  0.
↪0000  0.0000
chr1  111  10.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  10.0000  0.
↪0000  0.0000
chr1  121  12.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  13.0000  0.
↪0000  0.0000
```

Table : SV_SIMPLE.TSV file output column descriptions

Column	Description
template-name	The chromosome name.
position	The position in the reference chromosome.
proper-left	Count of properly paired left reads mapped in this location.
discordant-left	Count of discordantly paired left reads mapped in this location.
unmated-left	Count of unmated left reads mapped in this location.
proper-right	Count of properly paired right reads mapped in this location.
discordant-right	Count of discordantly paired right reads mapped in this location.
unmated-right	Count of unmated right reads mapped in this location.
not-paired	Count of single end reads mapped in this location.
unique	Count of unique mappings in this location.
ambiguous	Count of ambiguous mappings in this location.
n-count	The number of unknown bases on the reference for this location.

5.7.6 Discord command output file descriptions

The `discord` command uses clusters of discordant reads to find possible locations for structural variant breakends. The breakends are output in a VCF file called `discord_pairs.vcf.gz` using the `ALT` and `INFO` fields as defined in the VCF specification.

Note: RTG structural variant calls are stored in VCF format (version 4.1). For more information about the VCF format, refer to the specification online at: https://samtools.github.io/hts-specs/VCFv4.1.pdf

The following is an example of the VCF output of the `discord` command:

```
##fileformat=VCFv4.1
##fileDate=20120305
##source=RTGv2.5 build 9f7b8a5 (2012-03-05)
##CL=discord --template hst1 -o discord --readgroup-stats map/rgstats.tsv map/
↪alignments.bam
##RUN-ID=4157329b-edb9-419a-9129-44116e7a2195
##TEMPLATE-SDF-ID=4ecc9eb83e0ccec4
##reference=hst1
##contig=<ID="chr1",length=207900>
##INFO=<ID=CIPOS,Number=2,Type=Integer,Description="Confidence interval around POS␣
↪for imprecise variants">
##INFO=<ID=IMPRECISE,Number=0,Type=Flag,Description="Imprecise structural variation
↪">
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FILTER=<ID=INCONSISTENT,Description="Supporting reads are inconsistent as to␣
↪breakend location">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
#CHROM  POS  ID  REF  ALT  QUAL  FILTER  INFO  FORMAT  SAMPLE
chr1  50005  .  A  A[simulatedSequence1:52999[  .  PASS  IMPRECISE;SVTYPE=BND;
↪DP=506;CIPOS=0,0  GT  1/1
chr1  52999  .  G  ]simulatedSequence1:50005]G  .  PASS  IMPRECISE;SVTYPE=BND;
↪DP=506;CIPOS=0,0  GT  1/1
```

RTG adds custom fields to the VCF format to better account for some of its unique structural variant calling features, described in the tables below.

Table : RTG VCF file FILTER fields

Value	Description
PASS	Standard VCF "PASS", if breakend meets all the filtering criteria.
INCONSISTENT	Breakend with discordant read cluster that does not agree on the possible positions.

Table : RTG VCF file INFO fields

Value	Description
DP=<depth>	Indicates the number of discordant reads contributing to the cluster at this breakend.

If the --bed parameter is set the discord command also outputs a BED format file called discord_pairs.bed.gz containing the break-end regions. Any break-ends which do not have a PASS in the filter field of the VCF output will be preceded by the comment character in the BED file output.

The following is an example of the BED output from the discord command:

```
#Version v2.5 build 9f7b8a5 (2012-03-05), Discordance output 1
#CL  discord --bed --template hst1 -o discord --readgroup-stats map/rgstats.tsv␣
↪map/alignments.bam
#RUN-ID  4157329b-edb9-419a-9129-44116e7a2195
#chromosome  start  end  remote  count
chr1  50004  50004  remote:chr1:52998-52998  506
chr1  52998  52998  remote:chr1:50004-50004  506
```

The columns in the example are in BED file order:

1. Chromosome name
2. Start position in chromosome
3. End position in chromosome
4. Location of remote break-end matching this one
5. Count of discordant reads contributing to the break-end

5.7.7 Coverage command output file descriptions

The coverage command works out the coverage depth for a set of read alignments for a given reference. With default settings this will produce a BED format file containing the regions with a specific read depth. These regions are calculated by taking the read depth of each position as the average of itself and the read depths of the positions to the left and right of it out to the number specified with the --smoothing flag and then grouping the resulting values which have the same average read depth.

The following is an example of the BED output from the coverage command:

```
#Version v2.3.2 build 5c2ee18 (2011-10-05), Coverage BED output v1.0
#CL     coverage -o coverage-hslo-18-u -t hst1 hslo-18-u/alignments.bam
#RUN-ID c0561a1d-fb3b-4062-96ca-cad2cc3c476a
#sequence start   end     label           coverage
chr1      0       4       chr1        2
chr1      4       13      chr1        3
chr1      13      22      chr1        4
chr1      22      29      chr1        5
chr1      29      38      chr1        6
chr1      38      45      chr1        7
chr1      45      53      chr1        8
chr1      53      64      chr1        9
chr1      64      80      chr1        10
chr1      80      128     chr1        11
chr1      128     137     chr1        10
chr1      137     157     chr1        11
chr1      157     164     chr1        12
chr1      164     169     chr1        13
chr1      169     174     chr1        14
chr1      174     177     chr1        15
chr1      177     181     chr1        16
```

The columns in the example are in BED file order:

1. Chromosome name

2. Start position in chromosome
3. End position in chromosome
4. Name or label of the feature, this is generally the name of the chromosome or name of any BED features overlapping the coverage region
5. Depth of coverage for the range specified

When the `--per-region` flag is set, the `coverage` command will alter the criteria for outputting a BED record. Rather than defining regions having the same level of coverage, a BED record will be produced for each input BED region, containing the average coverage over that region.

When the `--bedgraph` flag is set, the `coverage` command will produce a BEDGRAPH format file with the regions calculated in the same way as for BED format output.

The following is an example of the BEDGRAPH output from the `coverage` command:

```
#Version v2.5.0 build 79d6626 (2011-10-05), Coverage BEDGRAPH output v1.0
#CL     coverage -o coverage-hslo-18-u -t hst1 hslo-18-u/alignments.bam --bedgraph
#RUN-ID 36c48ec4-52b7-48d5-b68c-71dce5dba129
track type=bedGraph name=coverage
chr1      0       4       2
chr1      4       13      3
chr1      13      22      4
chr1      22      29      5
chr1      29      38      6
chr1      38      45      7
chr1      45      53      8
chr1      53      64      9
chr1      64      80      10
chr1      80      128     11
chr1      128     137     10
chr1      137     157     11
chr1      157     164     12
chr1      164     169     13
chr1      169     174     14
chr1      174     177     15
chr1      177     181     16
```

The columns in the example are in BEDGRAPH file order:

1. Chromosome name
2. Start position in chromosome
3. End position in chromosome
4. Depth of coverage for the range specified

When the `--per-base` flag is set when running the `coverage` command will produce a tab separated value file with the coverage information for each individual base in the reference.

The following is an example of the TSV output from the `coverage` command:

```
#Version v2.3.2 build 5c2ee18 (2011-10-05), Coverage output v1.0
#CL  coverage -o coverage-hslo-18-u-per-base -t hst1 hslo-18-u/alignments.bam --
↪per-base
#RUN-ID 7798b1a5-2159-48e6-976e-86b4f8e98fa6
#sequence       position        unique-count    ambiguous-count   score
chr1            0               0               0                 0.00
chr1            1               0               0                 0.00
chr1            2               1               0                 1.00
chr1            3               1               0                 1.00
chr1            4               1               0                 1.00
chr1            5               1               0                 1.00
chr1            6               1               0                 1.00
```

```
chr1            7               2               0               2.00
chr1            8               2               0               2.00
chr1            9               2               1               2.50
```

Table : COVERAGE.TSV file output column descriptions

Column	Description
sequence	The chromosome name.
position	The position in the reference chromosome.
unique-count	The count of reads covering this position with IH equal to one.
ambiguous-count	The count of reads covering this position with IH greater than one.
score	The sum of one divided by the IH value for all the reads covering this position.

The coverage command produces a stats.tsv file with coverage statistics for each chromosome in the reference and the reference as a whole. The following is an example file:

```
#depth         breadth covered size    name
28.3238        0.9998  23766   23770   chr1
28.0013        0.9997  41447   41459   chr2
28.3151        0.9994  24930   24946   chr3
28.4955        0.9999  87734   87741   chr4
28.3822        0.9999  32600   32604   chr5
28.7159        0.9999  55042   55047   chr6
28.1109        0.9998  34887   34894   chr7
28.3305        0.9999  50025   50032   chr8
28.2229        0.9998  49627   49639   chr9
28.3817        0.9999  15912   15914   chr10
28.3569        0.9998  415970  416046  all sequences
```

Table : STATS.TSV column descriptions

Column	Description
depth	The average depth of coverage for the region where each base position is calculated as the sum of one divided by the IH of each read alignment which covers the position.
breadth	The fraction of the non-N region base positions which have a depth of one or greater.
covered	The number of non-N bases in the region which have a depth of one or greater.
size	The number of non-N bases in the region.
name	The name of the region, or “all sequences” for the entire reference.

For whole-genome coverage runs, the region names are each of the chromosomes. For exome data, or other targeted sequencing where a BED file was provided, the region names are obtained from the BED file.

The coverage command produces a levels.tsv file with some statistics about the coverage levels. The following is the start of an example file:

```
#coverage_level        count   %age    %cumulative
0       83      0.02    100.00
1       84      0.02    99.98
2       86      0.02    99.96
3       87      0.02    99.94
4       88      0.02    99.92
5       154     0.04    99.90
```

Table : LEVELS.TSV column descriptions

Column	Description
coverage_level	The coverage level.
count	The count of the number of bases at this coverage level.
%age	The percentage of the reference at this coverage level.
%cumulative	The percentage of the reference at this coverage level or higher.

5.7.8 Mapx output file description

The `mapx` command searches protein databases with translated nucleotide sequences. It reports matches filtered on a combination of percent identity, e-score, bit-score and alignment score. The matches are reported in an ASCII file called `alignments.tsv` with each match reported on a single line as a tab-separated value record.

The following results file is an example of the output produced by `mapx`:

```
#template-name frame read-id template-start template-end template-length read-
↪start read-end read-length template-protein read-protein alignment identical
↪%identical positive %positive mismatches raw-score bit-score e-score
gi|212691090|ref|ZP_03299218.1| +1      0       179     211     429     1       99␣
↪     100 nirqgsrtfgilcmpkasgnypallrvpgagvr       ␣
↪nirqgsrtfgifcmpkasgnypallrvpgaggr nirqgsrtfgi cmpkasgnypallrvpgag r       31     ␣
↪ 94      31      94      2       -162    67.0 2.3e-11
gi|255013538|ref|ZP_05285664.1| +1      0       176     208     428     1       99␣
↪     100 nvrpgsrtygilcmpkkegkypallrvpgagir       ␣
↪nirqgsrtfgifcmpkasgnypallrvpgaggr n+r gsrt+gi cmpk  g ypallrvpgag r       25     ␣
↪ 76      27      82      6       -136    57.0 2.3e-8
gi|260172804|ref|ZP_05759216.1| +1      0       185     217     435     1       99␣
↪     100 nicngsrtfgilcipkkpgkypallrvpgagvr       ␣
↪nirqgsrtfgifcmpkasgnypallrvpgaggr ni  gsrtfgi c+pk  g ypallrvpgag r       25     ␣
↪ 76      26      79      7       -129    54.3 1.5e-7
gi|224537502|ref|ZP_03678041.1| +1      0       162     194     414     1       99␣
↪     100 tdrwgsrfygvlcvpkkegkypallrvpgagir       ␣
↪nirqgsrtfgifcmpkasgnypallrvpgaggr r gsr +g+ c+pk  g ypallrvpgag r       21     ␣
↪64      24      73      9       -111    47.4 1.8e-5
```

The following table provides descriptions of each column in the `mapx` output file format, listed from left to right.

Table : ALIGNMENTS.TSV file output column descriptions

Column	Description
`template-name`	ID or description of protein (reference) with match.
`frame`	Denotes translation frame on forward (1,2,3) or reverse strand.
`read-id`	Numeric ID of read from SDF.
`template-start`	Start position of alignment on protein subject.
`template-end`	End position of alignment on protein subject.
`template-length`	Amino acid length of protein subject.
`read-start`	Start position of alignment on read nucleotide sequence.
`read-end`	End position of alignment on read nucleotide sequence.
`read-length`	Total nucleotide length of read.
`template-protein`	Amino acid sequence of aligned protein reference.
`read-protein`	Amino acid sequence of aligned translated read.
`alignment`	Amino acid alignment of match.
`identical`	Count of identities in alignment between reference and translated read.
`%identical`	Percent identity of match between reference and translated read, for exact matches only (global across translated read).
`positive`	Count of identical and similar amino acids in alignment between translated read and reference.
`%positive`	Percent similarity between reference and translated read, for exact and similar matches (global across translated read).
`mismatches`	Count of mismatches between reference and translated read.
`raw-score`	RTG alignment score (S); The alignment score is the negated sum of all single protein raw scores plus its penalties for gaps, which is the edit distance using one of the scoring matrices. Note that the RTG alignment score is the negated raw score of BLAST.
`bit-score`	Bit score is computed from the alignment score using the following formula: bit-score $= ((\lambda \times -S) - \ln(K))/\ln(2)$ where λ and K are taken from the matrix defaults [Blast pp.302-304] and S is the RTG alignment score.
`e-score`	e-score is computed from the alignment score using the following formula: e-score $= K \times m' \times n \times e^{(\lambda \times S)}$ n is the total length of the database. m' is the effective length of the query (read): $m' = \max(1, \text{querylength} + \lambda \times S/H)$

When the `--unmapped-reads` flag is set, unmapped reads are reported in an ASCII file called `unmapped.tsv` with each read reported on a single line as a tab-separated value record. Each read in the unmapped output has a character code indicating the reason the read was not mapped, with no code indicating that read had no matches.

Character codes for unmapped reads include:

Character	Description
`d`	Indicates that after alignment scores are calculated, the remaining hits were discarded because they exceeded the alignment score threshold (affected by `--max-alignment-score`).
`e`	Indicates that there were good scoring hits, but the arm was discarded because there were too many of these hits (affected by `--max-top-results`).
`f`	Indicates that there was a good hit which failed the percent identity threshold (affected by `--min-identity`).
`g`	Indicates that there was a good hit which failed the e-score threshold (affected by `--max-e-score`).
`h`	Indicates that there was a good hit which failed the bit score threshold (affected by `--min-bit-score`).

5.7.9 Species results file description

The `species` command estimates the proportion of taxa in a given set of BAM files. It does this by taking a set of BAM files which were mapped against a set of known genome sequences. The proportions are reported in a tab

separated ASCII file called `species.tsv` with each taxon reported on a separate line. The header lines include the command line and reference sets used.

In addition to the raw output, some basic diversity metrics are produced and output to the screen and to a file named `summary.txt`. The metrics included are:

:clist - Shannon (see http://en.wikipedia.org/wiki/Diversity_index#Shannon_index) - Pielou (see http://en.wikipedia.org/wiki/Species_evenness) - Inverse Simpson (see http://en.wikipedia.org/wiki/Diversity_index#Inverse_Simpson_index)

For an interactive graphical view of the `species` command output, an HTML5 report named `index.html`. Opening this shows the taxonomy and data on an interactive pie chart, with wedge sizes defined by either the abundance or DNA fraction (user selectable in the report).

The following results file is an example of the output produced by species:

```
#abundance abundance-low abundance-high DNA-fraction DNA-fraction-low DNA-fraction-
↪high confidence coverage-depth coverage-breadth reference-length mapped-reads␣
↪has-reference taxa-count taxon-id parent-id rank taxonomy-name
0.1693 0.1677 0.1710 0.06157 0.06097 0.06217 4.2e+02 0.4919 0.3915 1496992 20456.
↪00 Y 1 3 1 species Acholeplasma_laidlawii
0.1682 0.1671 0.1693 0.1384 0.1375 0.1393 6.6e+02 0.4892 0.3918 3389227 46059.50 Y␣
↪1 4 1 species Acidiphilium_cryptum
0.1680 0.1667 0.1692 0.09967 0.09891 0.1004 5.5e+02 0.4890 0.3887 2443540 33189.00␣
↪Y 1 6 1 species Acidothermus_cellulolyticus
0.1677 0.1668 0.1685 0.2301 0.2289 0.2313 8.7e+02 0.4877 0.3887 5650368 76543.00 Y␣
↪1 5 1 species Acidobacteria_bacterium
0.1646 0.1638 0.1655 0.2140 0.2129 0.2152 8.3e+02 0.4795 0.3886 5352772 71295.50 Y␣
↪1 7 1 species Acidovorax_avenae
0.1622 0.1614 0.1630 0.2562 0.2550 0.2574 9.2e+02 0.4721 0.3836 6503724 85288.00 Y␣
↪1 2 1 species Acaryochloris_marina
```

The following table provides descriptions of each column in the species output file format, listed from left to right.

Table : SPECIES.TSV file output column descriptions

Column	Description
abundance	Fraction of the individuals in the sample that belong to this taxon. The output file is sorted on this column.
abundance-low	Lower bound three standard deviations below the abundance.
abundance-high	Upper bound three standard deviations above the abundance.
DNA-fraction	Raw fraction of the DNA that maps to this taxon.
DNA-fraction-low	Lower bound three standard deviations below the DNA-fraction.
DNA-fraction-high	Upper bound three standard deviations above the DNA-fraction.
confidence	Confidence that this taxon is present in the sample. (Computed as the number of standard deviations away from the null hypotheses).
coverage-depth	The coverage depth of reads mapped to the taxon sequences, adjusted for the IH (number of output alignments) of the individual reads. (Zero if no reference sequences for the taxon).
coverage-breadth	The fraction of the taxon sequences covered by the reads. (Zero if no reference sequences for the taxon).
reference-length	The total length of the reference sequences, will be 0 if the taxon does not have associated reference sequences.
mapped-reads	The count of the reads which mapped to this taxon, adjusted for the IH (number of output alignments) of the individual reads.
has-reference	Y if the taxon has associated reference sequences, otherwise N.
taxa-count	The count of the number of taxa that are descendants of this taxon (including itself) and which are above the minimum confidence threshold.
taxon-id	The taxonomic ID for this result.
parent-id	The taxonomic ID of this results parent.
rank	The taxonomic rank associated with this result. This can be used to filter results into meaningful sets at different taxonomic ranks.
taxonomy-name	The taxonomic name for this result.

5.7.10 Similarity results file descriptions

The `similarity` command estimates how closely related a set of samples are to each other. It produces output in the form of a similarity matrix (`similarity.tsv`), a principal component analysis (`similarity.pca`) and two formats for phylogenetic trees showing relationships (`closest.tre` and `closest.xml`).

The similarity matrix file is a tab separated format file containing an $N \times N$ matrix of the matching *k*-mer counts, where *N* is the number of samples. An example of a `similarity.tsv` file:

```
#rtg similarity --unique-words -o similarity-out -I samples.txt -w 20 -s 15
F1_G_S_1M     F1_N_AN_1M      F1_O_BM_1M      F1_O_SP_1M      F1_O_TD_1M      F1_V_
↪PF_1M
F1_G_S_1M     8406725 4873    18425   13203   14746   10201
F1_N_AN_1M    4873    3551445 113310  117696  68118   245516
F1_O_BM_1M    18425   113310  6588017 579048  782011  152603
F1_O_SP_1M    13203   117696  579048  8583126 248654  161950
F1_O_TD_1M    14746   68118   782011  248654  9134232 79949
F1_V_PF_1M    10201   245516  152603  161950  79949   6000623
```

The principal component analysis file is a tab separated format file containing principal component groupings in three columns of real numbers, followed by the name of the sample. This can be turned into a 3D plot showing relationship groupings, by treating each line as a single point in three dimensional space. An example of a `similarity.pca` file:

```
0.0906        -0.1505 0.0471  F1_G_S_1M
-0.1207       -0.0610 -0.0348 F1_N_AN_1M
-0.0479       0.1889  -0.0119 F1_O_BM_1M
-0.0229       0.0893  0.0996  F1_O_SP_1M
0.0162        0.1375  -0.1384 F1_O_TD_1M
-0.1503       -0.0619 -0.0341 F1_V_PF_1M
```

The files containing the relationships as phylogenetic trees are in the Newick (`closest.tre`) and phyloXML (`closest.xml`) formats. For more information about the Newick tree format see http://en.wikipedia.org/wiki/Newick_format. For more information about the phyloXML format see http://www.phyloxml.org.

5.8 RTG JavaScript filtering API

The `vcffilter` command permits filtering VCF records via user-supplied JavaScript expressions or scripts containing JavaScript functions that operate on VCF records. The JavaScript environment has an API provided that enables convenient access to components of a VCF record in order to satisfy common use cases.

5.8.1 VCF record field access

This section describes the supported methods to access components of an individual VCF record. In the following descriptions, assume the input VCF contains the following excerpt (the full header has been omitted):

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA12877 NA12878
1 11259340 . G C,T . PASS DP=795;DPR=0.581;ABC=4.5 GT:DP 1/2:65 1/0:15
```

CHROM, POS, ID, REF, QUAL

Within the context of a `--keep-expr` or `record` function these variables will provide access to the String representation of the VCF column of the same name.

```
CHROM; // "1"
POS; // "11259340"
ID; // "."
REF; // "G"
QUAL; // "."
```

ALT, FILTER

Will retrieve an array of the values in the column.

```
ALT; // ["C", "T"]
FILTER; // ["PASS"]
```

INFO.{INFO_FIELD}

The values in the `INFO` field are accessible through properties on the `INFO` object indexed by `INFO ID`. These properties will be the string representation of info values with multiple values delimited with "`,`". Missing fields will be represented by "`.`". Assigning to these properties will update the VCF record. This will be undefined for fields not declared in the header.

```
INFO.DP; // "795"
INFO.ABC; // "4,5"
```

{SAMPLE_NAME}.{FORMAT_FIELD}

The JavaScript String prototype has been extended to allow access to the format fields for each sample. The string representation of values in the sample column are accessible as properties on the string matching the sample name named after the `FORMAT` field `ID`.

```
'NA12877'.GT; // "1/2"
'NA12878'.GT; // "1/0"
```

Note that these properties are only defined for fields that are declared in the VCF header (any that are not declared in the header will be undefined). See below for how to add new `INFO` or `FORMAT` field declarations.

5.8.2 VCF record modification

Most components of VCF records can be written or updated in a fairly natural manner by direct assignment in order to make modifications. For example:

```
ID = "rs23987382"; // Will change the ID value
QUAL = "50"; // Will change the QUAL value
FILTER = "FAIL"; // Will set the FILTER value
INFO.DPR = "0.01"; // Will change the value of the DPR info field
'NA12877'.DP = "10"; // Will change the DP field of the NA12877 sample
```

Other components of the VCF record (such as `CHROM`, `POS`, `REF`, and `ALT`) are considered immutable and can not currently be altered.

Direct assignment to `ID` and `FILTER` fields accept either a string containing semicolon separated values, or a list of values. For example:

```
ID = 'rs23987382;COSM3805';
ID = ['rs23987382', 'COSM3805'];
FILTER = 'BAZ;BANG';
FILTER = ['BAZ', 'BANG'];
```

Note that although the `FILTER` field returns an array when read, any changes made to this array directly are not reflected back into the VCF record.

Adding a filter to existing filters is a common operation and can be accomplished by the above assignment methods, for example by adding a value to the existing list and then setting the result:

```
var f = FILTER;
f.push('BOING');
FILTER = f;
```

However, since this is a little unwieldy, a convenience function called *add()* can be used (and may be chained):

```
FILTER.add('BOING');
FILTER.add(['BOING', 'DOING');
FILTER.add('BOING').add('DOING');
```

5.8.3 VCF header modification

Functions are provided that allow the addition of new `FILTER`, `INFO` and `FORMAT` fields to the header and records. It is recommended that the following functions only be used within the run-once portion of `--javascript`.

ensureFormatHeader(FORMAT_HEADER_STRING)

Add a new `FORMAT` field to the VCF if it is not already present. This will add a `FORMAT` declaration line to the header and define the corresponding accessor methods for use in record processing.

```
ensureFormatHeader('##FORMAT=<ID=GL,Number=G,Type=Float,' +
  'Description="Log_10 scaled genotype likelihoods.">');
```

ensureInfoHeader(INFO_HEADER_STRING)

Add a new `INFO` field to the VCF if it is not already present. This will add an `INFO` declaration line to the header and define the corresponding accessor methods for use in record processing.

```
ensureInfoHeader('##INFO=<ID=CT,Number=1,Type=Integer,' +
  'Description="Coverage threshold that was applied">');
```

ensureFilterHeader(FILTER_HEADER_STRING)

Add a new `FILTER` field to the VCF header if it is not already present. This will add an `FILTER` declaration line to the header.

```
ensureFilterHeader('##INFO=<ID=FAIL_VAL,' +
  'Description="Failed validation">');
```

5.8.4 Additional information and functions

SAMPLES

This variable contains an array of the sample names in the VCF header.

```
SAMPLES; // ['NA12877', 'NA12878']
```

print({STRING})

Writes the provided string to standard output.

```
print('The samples are: ' + SAMPLES);
```

checkMinVersion(RTG_MINIMUM_VERSION)

Checks the version of RTG that the script is running under, and exits with an error message if the version of RTG does not meet the minimum required version. This is useful when distributing scripts that make use of features that are not present in earlier versions of RTG.

```
checkMinVersion('3.9.2');
```

See also:

For javascript filtering usage and examples see *vcffilter*

5.9 Parallel processing approach

The comparison of genomic variation on a large scale in real time demands parallel processing capability. On a single server node, RTG automatically splits up compute-intensive commands like map and snp into multiple threads that run simultaneously (unless explicitly directed otherwise through the `-T` option).

To get a further reduction in wall clock time and optimize job execution on different size server nodes, a full read mapping and alignment run can be split into smaller jobs running in parallel on multiple servers.

How it works

With read mapping, very large numbers of reads may be aligned to a single reference genome. For example, 35 fold coverage of the human genome with 2×36 base pair data requires processing of 1,500 million reads.

Fortunately, multiple alignment files can be easily combined with other alignment files if the reference is held constant. Thus, the problem can easily be made parallel by breaking up the read data into multiple data sets and executing on separate compute nodes.

The steps for clustered read mappings with RTG are as follows:

Create multiple jobs of single end data, and start them on different nodes. By creating files with the individual jobs, these can be saved for repeated runs. This example shows the mapping of single end read data to a reference.

```
$ echo "rtg map -a 1 -b 0 -w 12 --start-read=0 --end-read=10000000 \
  -o ${RESULTS}/00 -i ${READ_DATA} -t ${REFERENCE}" > split-job-00
$ rsh host0 split-job-00
$ echo "rtg map -a 1 -b 0 -w 12 --start-read=10000000 --end-read=20000000 \
  -o ${RESULTS}/01 -i ${READ_DATA} -t ${REFERENCE}" > split-job-01
$ rsh host1 split-job-01
```

Repeat for as many segments of read data as required. This step can vary based on your configuration, but the basic idea is to create a command line script and then run it remotely. Store read, reference, and output results data in a central location for easier management. Shell variables that specify data location by URL are recommended to keep it all straight.

Run SNP caller to get variant detection results

```
$ rtg snp -t ${REFERENCE} -o ${RESULTS} ${RESULTS}/*/alignments.bam
```

With this pipeline, each job runs to completion on separate nodes mapping the reads against the same reference genome. Each node should have sufficient, dedicated memory. RTG will automatically use the available cores of the processor to run multi-threaded tasks.

Note: The map command does not require specific assignment of input files for left and right mate pair read data sets; this is handled automatically by the format command. The result is an SDF directory structure, with left and right SDF directories underneath a directory named by the `-o` parameter. Each of the left and right directories share a GUID and are identified as left or right arms respectively. The `map` command uses this information to verify that left and right arms are correct before processing.

Complete Genomics use case

Complete Genomics data has 70 lanes, typically. For clustered processing, plan one lane of mapping per SDF. If you have 7 machines available, map 10 on each machine resulting in 70 SDFs per machine, and 70 directories of BAM files per machine.

SDF read data is loaded into RAM during execution of the RTG `cgmap` command. For example, using human genomic data, ~1.28 billion reads (each with 70 bp of information) can be divided into chunks that can fit into memory by the `cgmap` command.

If you have multi-core server nodes available, the `cgmap` command will use multiple cores simultaneously. You can use the `-T` flag to adjust the number of cores used. Clustered processing dramatically reduces the wall clock time of the total job. At the end, the `snp` and `cnv` commands will accept multiple alignment files created by mapping runs at one time, where different sets of reads are mapped to the same reference.

5.10 Distribution Contents

The contents of the RTG distribution zip file should include:

- The RTG executable JAR file.
- RTG executable wrapper script.
- Example scripts and files.
- This operations manual.

- A release notes file and a readme file.

Some distributions also include an appropriate java runtime environment (JRE) for your operating system.

5.11 README.txt

For reference purposes, a copy of the distribution `README.txt` file follows:

```
=== RTG.VERSION ===

RTG software from Real Time Genomics includes tools for the processing
and analysis of plant, animal and human sequence data from high
throughput sequencing systems.  Product usage and administration is
described in the accompanying RTG Operations Manual.


Quick Start Instructions
========================

RTG software is delivered as a command-line Java application accessed
via a wrapper script that allows a user to customize initial memory
allocation and other configuration options. It is recommended that
these wrapper scripts be used rather than directly accessing the Java
JAR.

For individual use, follow these quick start instructions.

No-JRE:

  The no-JRE distribution does not include a Java Runtime Environment
  and instead uses the system-installed Java.  Ensure that at the
  command line you can enter "java -version" and that this command
  reports a java version of 1.7 or higher before proceeding with the
  steps below. This may require setting your PATH environment variable
  to include the location of an appropriate version of java.

Linux/MacOS X:

  Unzip the RTG distribution to the desired location.

  If your RTG distribution requires a license file (rtg-license.txt),
  copy the license file from Real Time Genomics into the RTG
  distribution directory.

  In a terminal, cd to the installation directory and test for success
  by entering "./rtg version"

  On MacOS X, depending on your operating system version and
  configuration regarding unsigned applications, you may encounter the
  error message:

  -bash: rtg: /usr/bin/env: bad interpreter: Operation not permitted

  If this occurs, you must clear the OS X quarantine attribute with
  the command:

  xattr -d com.apple.quarantine rtg

  The first time rtg is executed you will be prompted with some
  questions to customize your installation. Follow the prompts.
```

```
  Enter "./rtg help" for a list of rtg commands.  Help for any individual
  command is available using the --help flag, e.g.: "./rtg format --help"

  By default, RTG software scripts establish a memory space of 90% of
  the available RAM - this is automatically calculated.  One may
  override this limit in the rtg.cfg settings file or on a per-run
  basis by supplying RTG_MEM as an environment variable or as the
  first program argument, e.g.: "./rtg RTG_MEM=48g map"

  [OPTIONAL] If you will be running rtg on multiple machines and would
  like to customize settings on a per-machine basis, copy
  rtg.cfg to /etc/rtg.cfg, editing per-machine settings
  appropriately (requires root privileges).  An alternative that does
  not require root privileges is to copy rtg.example.cfg to
  rtg.HOSTNAME.cfg, editing per-machine settings appropriately, where
  HOSTNAME is the short host name output by the command "hostname -s"

Windows:

  Unzip the RTG distribution to the desired location.

  If your RTG distribution requires a license file (rtg-license.txt),
  copy the license file from Real Time Genomics into the RTG
  distribution directory.

  Test for success by entering "rtg version" at the command line.  The
  first time rtg is executed you will be prompted with some
  questions to customize your installation. Follow the prompts.

  Enter "rtg help" for a list of rtg commands.  Help for any individual
  command is available using the --help flag, e.g.: "rtg format --help"

  By default, RTG software scripts establish a memory space of 90% of
  the available RAM - this is automatically calculated.  One may
  override this limit by setting the RTG_MEM variable in the rtg.bat
  script or as an environment variable.


The scripts subdirectory contains demos, helper scripts, and example
configuration files, and comprehensive documentation is contained in
the RTG Operations Manual.

Using the above quick start installation steps, an individual can
execute RTG software in a remote computing environment without the
need to establish root privileges.  Include the necessary data files
in directories within the workspace and upload the entire workspace to
the remote system (either stand-alone or cluster).

For data center deployment and instructions for editing scripts,
please consult the Administration chapter of the RTG Operations Manual.

A discussion group is now available for general questions, tips, and other
discussions. It may be viewed or joined at:
https://groups.google.com/a/realtimegenomics.com/forum/#!forum/rtg-users

To be informed of new software releases, subscribe to the low-traffic
rtg-announce group at:
https://groups.google.com/a/realtimegenomics.com/forum/#!forum/rtg-announce

Citing RTG
==========

John G. Cleary, Ross Braithwaite, Kurt Gaastra, Brian S. Hilbush,
```

```
Stuart Inglis, Sean A. Irvine, Alan Jackson, Richard Littin, Sahar
Nohzadeh-Malakshah, Mehul Rathod, David Ware, Len Trigg, and Francisco
M. De La Vega. "Joint Variant and De Novo Mutation Identification on
Pedigrees from High-Throughput Sequencing Data." Journal of
Computational Biology. June 2014, 21(6):
405-419. doi:10.1089/cmb.2014.0029.

Terms of Use
============

This proprietary software program is the property of Real Time
Genomics.  All use of this software program is subject to the
terms of an applicable end user license agreement.

Patents
=======

US: 7,640,256, 13/129,329, 13/681,046, 13/681,215, 13/848,653,
13/925,704, 14/015,295, 13/971,654, 13/971,630, 14/564,810
UK: 1222923.3, 1222921.7, 1304502.6, 1311209.9, 1314888.7, 1314908.3
New Zealand: 626777, 626783, 615491, 614897, 614560
Australia: 2005255348, Singapore: 128254
Other patents pending


Third Party Software Used
=========================

RTG software uses the open source htsjdk library
(https://github.com/samtools/htsjdk) for reading and writing SAM
files, under the terms of following license:

The MIT License

Copyright (c) 2009 The Broad Institute

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

-------------------------

RTG software uses the bzip2 library included in the open source Ant project
(http://ant.apache.org/) for decompressing bzip2 format files, under the
following license:

Copyright 1999-2010 The Apache Software Foundation

Licensed under the Apache License, Version 2.0 (the "License"); you may not
```

```
use this file except in compliance with the License. You may obtain a copy of
the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed
under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND, either express or implied. See the License for the
specific language governing permissions and limitations under the License.

-------------------------

RTG Software uses a modified version of
java/util/zip/GZIPInputStream.java (available in the accompanying
gzipfix.jar) from OpenJDK 7 under the terms of the following license:

This code is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License version 2 only, as
published by the Free Software Foundation.  Oracle designates this
particular file as subject to the "Classpath" exception as provided
by Oracle in the LICENSE file that accompanied this code.

This code is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
version 2 for more details (a copy is included in the LICENSE file that
accompanied this code).

You should have received a copy of the GNU General Public License version
2 along with this work; if not, write to the Free Software Foundation,
Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.

Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
or visit http://www.oracle.com if you need additional information or have
any questions.

-------------------------

RTG Software uses hierarchical data visualization software from
http://sourceforge.net/projects/krona/ under the terms of the
following license:

Copyright (c) 2011, Battelle National Biodefense Institute (BNBI);
all rights reserved. Authored by: Brian Ondov, Nicholas Bergman, and
Adam Phillippy

This Software was prepared for the Department of Homeland Security
(DHS) by the Battelle National Biodefense Institute, LLC (BNBI) as
part of contract HSHQDC-07-C-00020 to manage and operate the National
Biodefense Analysis and Countermeasures Center (NBACC), a Federally
Funded Research and Development Center.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.
```

```
* Neither the name of the Battelle National Biodefense Institute nor
  the names of its contributors may be used to endorse or promote
  products derived from this software without specific prior written
  permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

5.12 Notice

Real Time Genomics does not assume any liability arising out of the application or use of any software described herein. Further, Real Time Genomics does not convey any license under its patent, trademark, copyright, or common-law rights nor the similar rights of others.

Real Time Genomics reserves the right to make any changes in any processes, products, or parts thereof, described herein, without notice. While every effort has been made to make this guide as complete and accurate as possible as of the publication date, no warranty of fitness is implied.

Illumina, Solexa, Complete Genomics, Ion Torrent, Roche, ABI, Life Technologies, and PacBio are registered trademarks and all other brands referenced in this document are the property of their respective owners.