RTG Tools Operations Manual

Release 3.8

Real Time Genomics

May 15, 2017

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.

The RTG Tools platform provides a subset of the functionality available from the full suite of functions for analyzing and manipulating variant call results. These utilities can be used to perform a variety of tasks such as:

- **Accuracy Evaluation** – Compare called variants to a set of known variants to find specificity and sensitivity, check mendelian consistency for the variants from a family, finding basic variant statistics for a set of calls.
- **Result Filtering** – Find a subset of variants that match a given set of filtering criteria, extracting only the variant information required for a specific task.
- **Variant Set Manipulation** – Merging multiple sets of variant results together, adding additional annotation information to existing variants.

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>
```

See also:

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

1.3 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).

1.3.1 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.

1.3.2 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*.

1.4 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

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 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*

2.3.3 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*

2.3.4 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*

2.3.5 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-trim-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

2.4 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
```

2.4.1 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*

2.4.2 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*

2.4.3 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 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 new output VCF will contain all the existing variants and samples with a new column for the new sample.

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*

2.4.4 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*

2.4.5 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*

2.4.6 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		
`-h`	`--help`	Prints help on command-line flag usage.
`-Z`	`--no-gzip`	Set this flag to create the VCF output file without compression.
	`--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.
	`--sex=SEX`	Specifies the sex of the 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.

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 `--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*

2.4.7 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		
`-h`	`--help`	Print help on command-line flag usage.
`-Z`	`--no-gzip`	Do not gzip the output.
	`--num-crossovers=FLOAT`	Likelihood of extra crossovers per chromosome (Default is 0.01)
	`--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*

2.4.8 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*

2.5 Utility Commands

2.5.1 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

2.5.2 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*

2.5.3 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 'chr1:10000+10'
```

Parameters:

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

Filtering		
	STRING+	Specifies the region to display. The format is one of <sequence_name>, <sequence_name>:start-end or <sequence_name>:start+length. 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*

2.5.4 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=STRING	The region of interest to display. The format is one of <sequence_name>, <sequence_name>:start-end or <sequence_name>:start+length.
	--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-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.
	--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*

2.5.5 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, *sdf2fasta*, *sdf2fastq*, *sdfstats*

2.5.6 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*

2.5.7 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`	Set to use sequence id instead of sequence name in region flag (0-based).
	`STRING+`	Specifies the region to display. The format is one of <sequence_name>, <sequence_name>:start-end or <sequence_name>:start+length. 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*

2.5.8 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 or '-' to read from standard input.
	--output=FILE	If set, output annotated calls to this VCF file.
	--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.
	--no-index	Do not produce indexes for output files.

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*.

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, you can end up with situations where calls are not available for every member of the family. Under normal circumstances these will be reported as an allele count constraint violation. It is possible 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*

2.5.9 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.

See also:

snp, *family*, *somatic*, *vcffilter*, *vcfmerge*, *vcfsubset*

2.5.10 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		
`-a`	`--add-header=STRING`	Add the supplied text to the output VCF header. May be specified 0 or more times.
`-o`	`--output=FILE`	Output VCF file. Use '-' to write to standard output.
	`FILE+`	Input VCF files to merge. Must be specified 1 or more times.

Utility		
-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.
	--no-index	Do not produce indexes for output files.
	--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` parameter. 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. The `--preserve-formats` flag prevents this loss of information by refusing to merge the records (separate records will be output).

See also:

snp, *family*, *population*, *somatic*, *vcffilter*, *vcfsubset*, *bgzip*, *index*

2.5.11 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.

Syntax:

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

Example:

```
$ rtg vcffilter -i snps.vcf.gz -o snps_cov5.vcf.gz -d 5
```

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=STRING`	If set, only read VCF records within the specified range. The format is one of <sequence_name>, <sequence_name>:start-end or <sequence_name>:start+length.
	`--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`	Set a window length in which multiple called variants are discarded.
	`--exclude-bed=FILE`	Set to discard all variants within the regions contained in the BED file.
	`--exclude-vcf=FILE`	Set to discard all variants that overlap with the ones in this VCF file.
	`--include-bed=FILE`	Set to only keep variants within the regions contained in the BED file.
	`--include-vcf=FILE`	Set to only keep variants that overlap with the ones in this VCF file.
`-j`	`--javascript=STRING`	Specify filtering functions in javascript. May be either an expression or a file name. May be specified 0 or more times. *See Examples*
`-e`	`--keep-expr=STRING`	Set to only keep variants for which this expression evaluates to true. *See Examples*
`-k`	`--keep-filter=STRING`	Set to only keep variants with this FILTER tag. May be specified 0 or more times, or as a comma separated list.
`-K`	`--keep-info=STRING`	Set to only keep variants with this INFO tag. May be specified 0 or more times, or as a comma separated list.
`-C`	`--max-combined-read-depth=INT`	Set the maximum allowed combined read depth.
`-Q`	`--max-quality=FLOAT`	Set the maximum allowed quality.
`-c`	`--min-combined-read-depth=INT`	Set the minimum allowed combined read depth.
`-q`	`--min-quality=FLOAT`	Set the minimum allowed quality.
	`--non-snps-only`	Set to output MNPs and INDELs only.
`-r`	`--remove-filter=STRING`	Set to remove variants with this FILTER tag. May be specified 0 or more times, or as a comma separated list.
`-R`	`--remove-info=STRING`	Set to remove variants with this INFO tag. May be specified 0 or more times, or as a comma separated list.
	`--remove-overlapping`	Set to remove records that overlap with previous records.
	`--snps-only`	Set to output simple SNPs only.

Filtering (Sample based)		
-A	--max-ambiguity-ratio=FLOAT	Set the maximum allowed ambiguity ratio.
	--max-avr-score=FLOAT	Set the maximum allowed AVR score.
	--max-denovo-score=FLOAT	Set the maximum allowed de novo score.
-G	--max-genotype-quality=FLOAT	Set the maximum allowed genotype quality.
-D	--max-read-depth=INT	Set the maximum allowed sample read depth.
	--min-avr-score=FLOAT	Set the minimum allowed AVR score.
	--min-denovo-score=FLOAT	Set the minimum allowed de novo score.
-g	--min-genotype-quality=FLOAT	Set the minimum allowed genotype quality.
-d	--min-read-depth=INT	Set the minimum allowed sample read depth.
	--remove-all-same-as-ref	Set to remove records where all the samples are same as the reference.
	--remove-hom	Remove where sample is homozygous.
	--remove-same-as-ref	Set to remove variants where the sample is the same as reference.

Reporting		
	--clear-failed-samples	Instead of removing failed records set the sample GT fields to missing.
	--fail=STRING	Instead of removing failed records set their filter field to the provided value.

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.
	--no-index	Do not produce indexes for output files.

Usage:

Use `vcffilter` to get a subset of the results from variant calling based on the filtering criteria supplied by the filter flags. 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 by setting their GT field to missing by using the `--clear-failed-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.

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})"
```

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 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 = sample.DP / maxAltDepth;
    }
  });
}
```

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*, *vcfsubset*

2.5.12 vcfannotate

Synopsis:

Used to add annotations to a VCF file, either to the VCF `ID` field, or as a VCF `INFO` 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		
	`--bed-ids=FILE`	Specifies a file in BED format containing variant ids in the name column to be added to the VCF id field. May be specified 0 or more times.
	`--bed-info=FILE`	Specifies a file in BED format containing annotations in the name column to be added to the VCF info field. May be specified 0 or more times.
	`--fill-an-ac`	Set to add or update the AN and AC info fields to the VCF.
	`--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`	Relabel samples according to `old-name new-name` pairs in specified file. If only a single sample needs to be relabelled then a construct like `<(echo old-name new-name)` can be used.
	`--vcf-ids=FILE`	Specifies a file in VCF format containing variant ids to be added to the VCF id field. May be specified 0 or more times.

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.
	`--no-index`	Do not produce indexes for output files.

Usage:

Use `vcfannotate` to add text annotations to variants that fall within ranges specified in a BED or VCF file. The annotations from the BED file are added as an `INFO` field in the output VCF file.

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.

See also:

snp, *family*, *somatic*, *population*, *vcffilter*, *vcfsubset*

2.5.13 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.
	`--no-index`	Do not produce indexes for output files.

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*

2.5.14 vcfeval

Synopsis:

Use the `vcfeval` command to evaluate called variants for agreement with a known baseline variant set.

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	The VCF file containing baseline variants. For example, these may be the variants that were used to generate a synthetic sample, 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.
	--bed-regions=FILE	If set, only read VCF records that overlap the ranges contained in the specified BED file.
	--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.
-c	--calls=FILE	The VCF file containing called variants.
-o	--output=DIR	The name of the output directory.
	--region=STRING	If set, only read VCF records that overlap the specified region. The format is one of <template_name>, <template_name>:start-end or <template_name>:start+length
-t	--template=SDF	The reference SDF on which the variants were called.

Filtering		
	--all-records	Set to use all records regardless of filters. Default is to only process records where FILTER is . or PASS.
	--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	Set the name of the sample to select. Use the form <baseline_sample>,<calls_sample> to select different sample names for baseline and calls. (Required when using multi-sample VCF files).
	--squash-ploidy	Treat heterozygous variants as homozygous ALT in both baseline and calls.

Reporting		
	--output-mode	Output reporting mode (Must be one of [split, annotate, combine, ga4gh, roc-only]). (Default is split).
-O	--sort-order=STRING	Set the order in which to sort the ROC scores so that "good" scores come before "bad" scores. (Must be one of [ascending, descending]). (Default is descending).
-f	--vcf-score-field=STRING	Set the VCF format field to sort the ROC using. Also valid are "QUAL" or "INFO.<name>" to select the named VCF 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.

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 call variants which were represented as SNPs. This file includes a subset of accuracy metrics, as the computation of some metrics is not meaningful on a subset of the data where representation may differ between the baseline and the call.
- `non_snp_roc.tsv` – contains ROC data derived from only those call variants which were not represented as SNPs. As above, not all metrics are computed for this file.

A common desire is to perform analysis separately for SNPs vs 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. In the `snp_roc.tsv` and `non_snp_roc.tsv` files above, the representation used by the called variants is used to classify the variant type, as it is the called variants which contain the scores which are used to determine the ranking for ROC curves. Thus the type-specific ROC data files should not be considered portable from one caller to another.

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

---

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 repre-

sentational 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

See also:

snp, *popsim*, *samplesim*, *childsim*, *rocplot*

2.5.15 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-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*

2.5.16 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		
	--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. In addition, it is possible to generate a simple pedigree visualization, using GraphViz, which can be saved to PNG or PDF. For

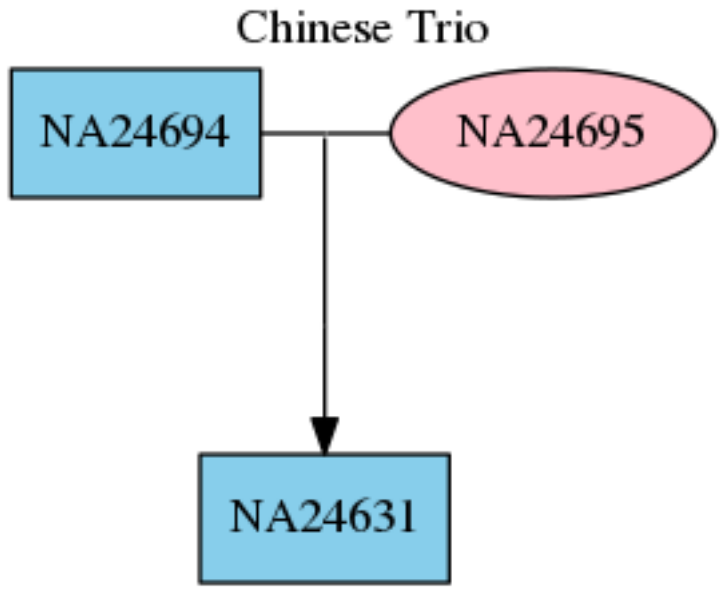
example, with the following `chinese-trio.ped`:

```
#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.

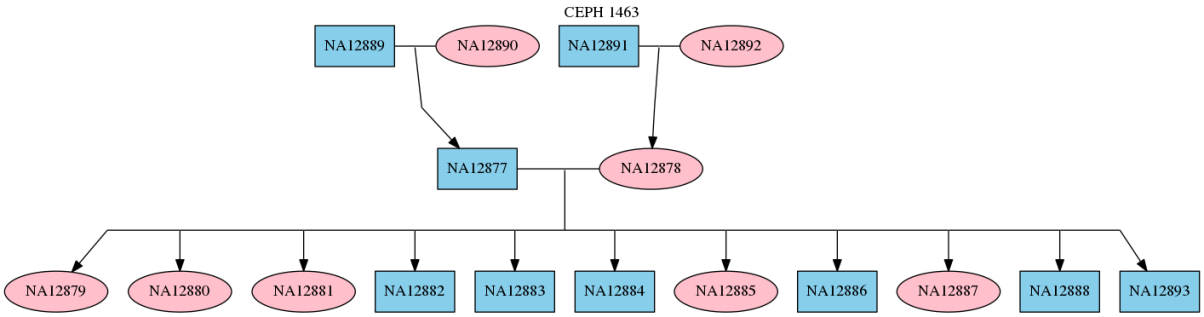


For more information about the PED file format see *Pedigree PED input file format*.

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:



See also:

family, *population*, *pedfilter*

2.5.17 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.
	`--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` 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.

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.

Some quick tips for the interactive GUI:

- Select regions within the graph to zoom in. Right click to bring up a context menu that allows resetting the zoom.
- 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 clear 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.
- 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.
- Each ROC curve can be shown/hidden, renamed, and reordered in it's widget area on the right hand side of the UI.
- Right-clicking within the ROC widget area allows permanently removing that ROC curve.
- Each ROC 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.
- The 'Cmd' button will open a message window that contains a command-line equivalent to the currently displayed set of ROC 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.

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

---

See also:

readsimeval, *vcfeval*

2.5.18 version

Synopsis:

The RTG version display utility.

Syntax:

```
$ rtg version
```

Example:

```
$ rtg version

Product: RTG Core 3.5
Core Version: 4586490 2015-12-04
RAM: 3.5GB of 3.8GB RAM can be used by RTG (91%)
CPU: Defaulting to 4 of 4 available processors (100%)
License: Expires on 2016-03-30
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, 2014
```

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*

2.5.19 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*

2.5.20 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

ADMINISTRATION & CAPACITY PLANNING

3.1 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.

3.2 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.

3.3 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.

3.4 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*.

3.5 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.

3.5.1 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.

3.5.2 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.

3.5.3 Advanced 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
```

CHAPTER

FOUR

APPENDIX

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.4.1 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"
REF; // "G"
```

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"

INFO.DPR = "0.01"; // Will change the value of the DPR info field
```

{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` These properties can be assigned in order to make modifications. This will be undefined for fields not declared in the header.

```
'NA12877'.GT; // "1/2"
'NA12878'.GT; // "1/0"
'NA12877'.DP = "10"; // Will change the DP field of the NA12877 sample
```

4.4.2 VCF header modification

Functions are provided that allow the addition of new `INFO` or `FORMAT` fields to the header and records. It is recommended that the following functions only be used within the run-once portion of `--javascript`. They may be called on every record, but this will be slow.

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">');
```

4.4.3 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);
```

See also:

For javascript filtering usage and examples see *vcffilter*

4.5 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.

4.6 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
```

4.7 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.