CGA™ Tools User Guide

Software Version 1.8

UG_CTA-1.8-01

Table of Contents

Preface

Complete Genomics Analysis Tools (CGA™ Tools) is an open source project to provide tools for downstream analysis of Complete Genomics data. This document describes how to install and use the tools, and provides information on the underlying algorithms.

Conventions

This document uses the following notational conventions:

Notation	Description
italic	A field name from a data file. For example, the *varType* field in the variations data file indicates the type of variation identified between the assembled genome and the reference genome. Also used to indicate the values found in data files. For example, *ChromosomeName* indicates that the value found in the data file is the name of a given chromosome.
bold_italic	A file name from the data package. For example, each package contains the file ***manifest.all***.
42$ echo hello world	Indicates that the reader type "echo hello world" (without quotes) at the command prompt. The steps are numbered (bold number before the prompt) sequentially through the document

Complete Genomics Data and Data Structure Requirements

Complete Genomics, Inc. delivers complete genome sequencing data to customers and collaborators. The data include sequence reads, their mappings to a reference human genome, and variations detected against the reference human genome. This document describes tools developed to analyze this data.

Data Structure Requirements

Several of the CGA Tools utilities, such as evidence2sam, calldiff, and generatemastervar, require input files to be located in the directory hierarchy in which the data package was originally delivered by Complete Genomics. To ensure that CGA Tools utilities function properly, you need to maintain the original directory hierarchy for each genome that is to be analyzed. Figure 1 and Figure 2 show the standard top level and ASM directory hierarchies for genomes sequenced using Complete Genomics Standard Sequencing Service.

Figure 1: Top-Level Data Structure

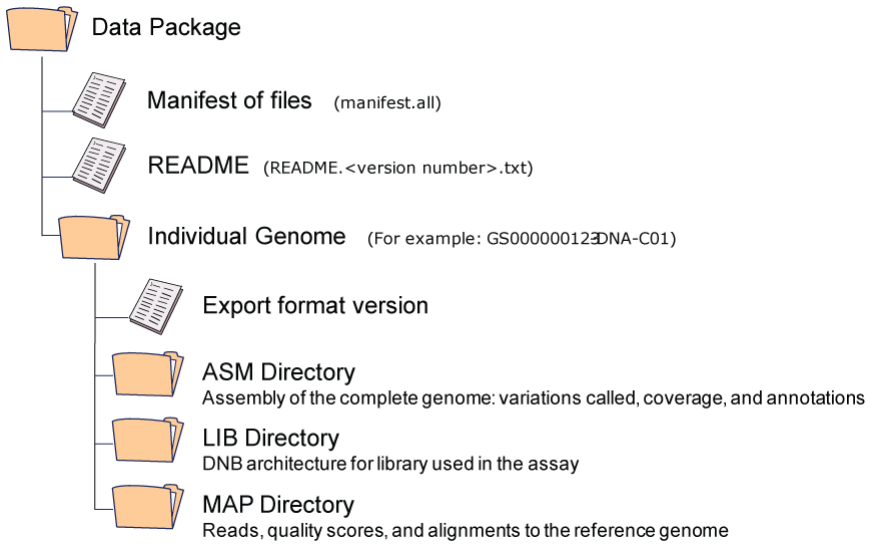
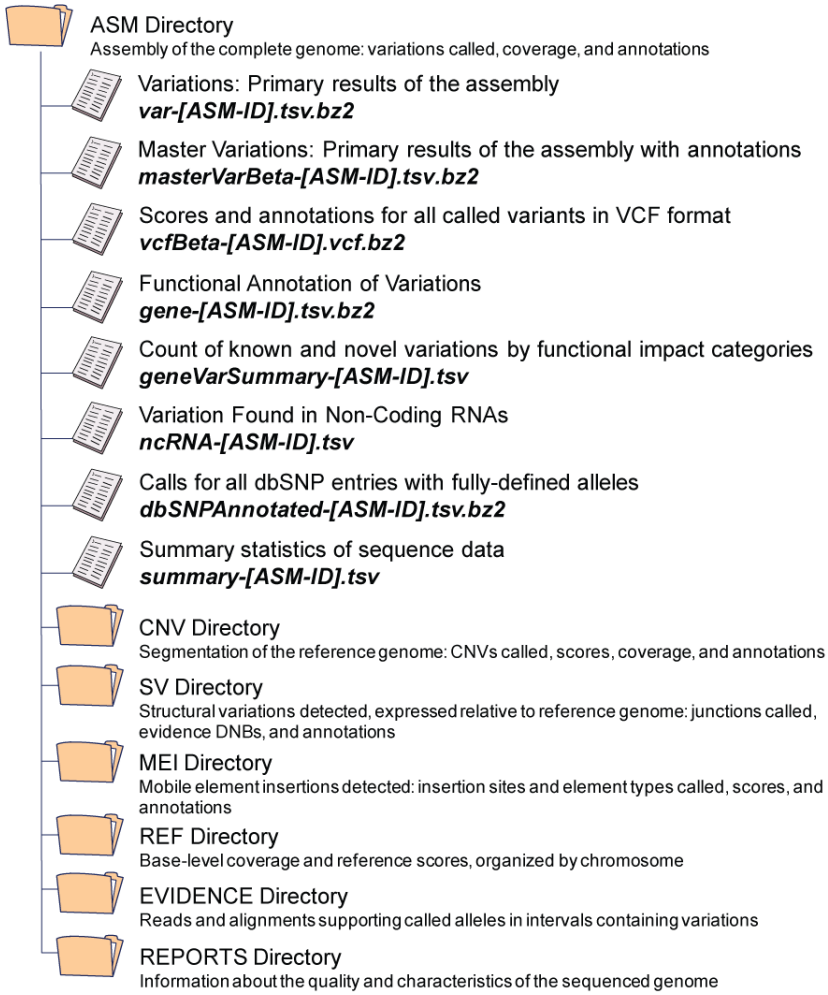


Figure 2: Standard ASM Directory Hierarchy



CGA Tools Documents

This document is part of a set that describe aspects of CGA Tools and the Complete Genomics human genome data. These documents are available from the Complete Genomics website (except where noted):

- *Complete Genomics Data File Formats* — The organization and content of the format used to deliver Complete Genomics human genome data, available for the standard sequencing service and the cancer sequencing service.
- *Analysis Pipeline Release Notes* — New features and enhancements listed by release.
- *Complete Genomics Managing Data FAQ* — Answers to questions about preparing to receive the hard drives of data.
- *Complete Genomics Small Variants FAQ* — Answers to frequently asked questions about Complete Genomics variation data.
- *Complete Genomics CNV, SV, and MEI FAQ* — Answers to frequently asked questions about Complete Genomics variation data.
- *Complete Genomics Small Variant Score Calibration Methods* — Methods used to calibrate Complete Genomics small variant quality scores to absolute error rate. Error calibration is based on replicate experiments conducted by Complete Genomics at various levels of coverage. These methods and the related calibration files are used by the CGA Tools mkvcf (when producing calibrated scores) and calldiff (with the SomaticOutput) tools.

References

CGA Tools requires users to be familiar with Unix-like commands. Here are some basic resources to get you started or enhance your skills:

- A Brief UNIX Shell Comparison: http://www.thewellroundedgeek.com/2007/04/brief-unix-shell-comparison.html
- An alphabetical list of basic Unix/Linux commands with thorough examples: http://www.math.utah.edu/lab/unix/unix-commands.html
- A complete list of Linux commands and their options: http://www.oreillynet.com/linux/cmd/
- Python interpreter: This open source programming language gives users of Complete Genomics data a powerful analysis platform. You won't need to interact with Python for this installation, but you may find that you want to use it to expand your data analysis capabilities. Here's where to get started with Python: http://docs.python.org/tutorial/.

Recompiling CGA Tools requires users to have installed several Unix utilities. See the resources below for downloads and more information.

- CMake: An extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner. Here's an overview of the utility and its operation: http://www.cmake.org/Wiki/CMake.
- Boost C++ libraries: This freely downloaded, peer-reviewed set of C++ libraries provides the core C++ functionality on which the CGA Tools source code is built. You will not need to interact with Boost other than to make sure it is installed and available during the CGA Tools installation process. Here's where to find out more: http://www.boost.org/users/faq.html.

  For detailed documentation and installation instructions, see:

  www.boost.org/doc/libs/1_46_1/more/getting_started/unix-variants.html

Complete Genomics provides the following Baseline data:

- Sample Sequence Data: Complete Genomics release of public genome sequences: www.completegenomics.com/sequence-data/download-data/.

- Baseline Genome Set — The data used to generate the baseline genome set is comprised of 52 unrelated genomes from the Complete Genomics Diversity Panel. The following summaries are available of this data:
  - CNV Baseline Genome Dataset: Summary of the underlying data and normalization constants for each of the CNV baseline genomes. The accompanying *Data Format Description* document provides the identifiers for each genome in the CNV baseline set and describes the data file format for the CNV baseline genome composite file. Available from the Complete Genomics FTP site. [ftp://ftp2.completegenomics.com/Baseline_Genome_Set/CNVBaseline]
  - SV Baseline Genome Dataset: Summary of the detected junctions and their frequencies across the SV baseline set. The accompanying *Data Format Description* document provides the identifiers for each genome in the SV baseline set and describes the data file format for the SV baseline genome composite file. Available from the Complete Genomics FTP site. [ftp://ftp2.completegenomics.com/Baseline_Genome_Set/SVBaseline]

CGA Tools draw on the following industry standards and publicly available data:

- SAM — The Sequence Alignment/Map format is a generic format for storing large nucleotide sequence alignments. This CGA Tools description is based on the SAM Format Specification "Sequence Alignment/Map (SAM) Format," Version 0.1.2-draft (August 20, 2009). [http://samtools.sourceforge.net/SAM1.pdf]
- NCBI RefSeq alignment data — Reference assembly and alignment data. This data is available here:

  ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/ARCHIVE/

  Select the appropriate Annotation Build, then navigate to:

  ```
  mapview/seq_gene.md.gz
  ```

- Reference Sequence (RefSeq) Information — Functional impact of variants in the coding regions of genes is determined using RefSeq annotation data. Refer to the following sources:
  - RefSeq — Database of reference sequences annotations of DNA. [www.ncbi.nlm.nih.gov/refseq/]
  - Release Notes — information on a given annotation build. [www.ncbi.nlm.nih.gov/genome/guide/human/release_notes.html]

Many of the CGA Tools are packaged in a Galaxy tool repository and can be installed directly into a Galaxy instance for genome sequencing data analysis:

- Galaxy Project — an open, web-based platform for data intensive biomedical research.

Installing CGA™ Tools

Overview and Requirements

CGA Tools is an open source project to provide tools for downstream analysis of Complete Genomics data. The CGA Tools source code, as well as precompiled binaries for Linux and Mac OS X, are available from SourceForge. The source code can be compiled and run on Linux, Unix, or Mac OS X systems. CGA Tools binaries are also available on the Galaxy platform though the Galaxy main tool shed.

Install Processes in a Nutshell

Install CGA Tools from a Binary Distribution

You can install CGA Tools software on 64-bit Linux or Mac OS X systems from pre-compiled binary distributions provided by Complete Genomics. This document provides detailed instructions for these high-level steps:

1. Install CGA Tools from a Binary Distribution: Mac OS X or Linux.
2. Obtain a copy of the reference human genome formatted for use with CGA Tools.
3. Obtain additional source files for use with CGA Tools.

Install CGA Tools within a Galaxy Environment

Alternatively, you can install the Linux or Mac OS X versions of CGA Tools within a Galaxy environment running on a local computer or a dedicated server. The process includes these high-level steps:

1. Install CGA Tools from the Galaxy main tool shed.
2. Download human reference and additional auxiliary files for use with CGA Tools.

Building CGA Tools from Source Code

You can also build CGA Tools from source code. This document provides detailed instructions for these high-level steps:

1. Prepare your environment.
2. Verify that you have the required third-party software.
3. Build and install CMake.
4. Build and install Boost.
5. Use CMake to build and install CGA Tools.
6. Obtain a copy of the reference human genome formatted for use with CGA Tools.
7. Obtain additional source files for use with CGA Tools.

User Requirements

- Command-line Unix skills

System Requirements

- CentOS 5.1 (64-bit) Linux or equivalent.

  This package works on 32-bit or 64-bit Linux and Unix-like systems, including Mac OS X (via the terminal program). However, Complete Genomics has formally tested only the CentOS 64-bit environment. It has not been tested on Microsoft Windows.

- 8 GB RAM available.

  Note that some of the CGA Tools commands, such as join, require more memory proportionate to the size of the input file.

- Disk usage proportional to the data.

  Typically, for a standard coverage genomes, you will need 30-50 GB disk space per genome when working with Complete Genomics variation and evidence data and reference data. If you also need to use the reads and initial mappings, your Complete Genomics data may be 300 GB or more per standard coverage genome. Depending on what you intend to do with the data, you will need even more disk space than this: for example, if you intend to convert the initial mapping data from a standard coverage genome to SAM and BAM format, you will need more than 2 TB of space, some of which is taken up by intermediate files (the SAM files) that can be deleted after the conversion.

Software Requirements to Build CGA Tools from Source Code

- GCC C++ compiler, version 4.1.2 or later
- CMake version 2.8.1 or later
- Boost version 1.46.1
- Python version 2.4.3 or later (only required if running test cases)
- Doxygen version 1.6.2 or later (only required if generating API docs)

Note: You may have older versions of some of these software programs installed, or they may be installed in a manner incompatible for the CGA Tools build process. If you encounter error messages during the CGA Tools build process, we recommend you download and install fresh instances of the recommended CMake and Boost versions. Consider installing them in a private directory separate from the other version on your computer.

Installing CGA Tools from a Binary Distribution

CGA Tools is available as a pre-compiled binary distribution for Mac OS X and 64-bit Linux. Instructions for preparing and installing the binary distribution are included for each platform:

- Mac OS X: Installing CGA Tools from a Binary Distribution
- Linux: Installing CGA Tools from a Binary Distribution

Mac OS X: Installing CGA Tools from a Binary Distribution

1. Create a directory structure.

   We recommend the following directory structure to support the installation and use of CGA Tools. This structure assumes your username is “complete” and all Complete Genomics data and applications are installed by user “complete”.

Content	Directory Structure
CGA Tools executable **Note:** If this directory does not already exist, it requires root privileges to create.	`/usr/local/bin`
CGA Tools documentation	`/Users/complete/cgatools_docs`
Your Complete Genomics human genome data files	`/Users/complete/data`
Reference human genome sequence files	`/Users/complete/data/ref`

The installation instructions assume this directory structure. If you choose to create a variation on this structure, make sure to adapt the installation instructions below to your structure.

2. Validate the system path.

   The default system $PATH includes /usr/local/bin. If you installed in an alternate location, be sure the directory containing the CGA Tools executable is in your $PATH. Consult your shell documentation for how to set the $PATH.

3. Download the binary.

   The current CGA Tools binary distribution is available from SourceForge (http://cgatools.sourceforge.net). The file is named as follows, where the X's are replaced by the current version number:

   ```
   cgatools-X.X.X.X-MacOSX_binary-x86_64.tar.gz
   ```

4. Untar the tarball by double clicking the filename in the Finder, or use the following command:

   ```
   1$ tar -xzf cgatools-X.X.X.X-MacOSX_binary-x86_64.tar.gz
   ```

   This command creates a directory including CGA Tools executable and documents:

   - Executable: `/bin/cgatools`
   - Documentation: `/share/cgatools-X.X.X/cgatools-X.X.X-docs/index.html`

5. Copy the executable into your binary directory:

   ```
   2$ sudo cp cgatools /usr/local/bin
   ```

6. Copy the documentation into your documentation directory:

   ```
   3$ cp –r cgatools-X.X.X.X-MacOSX_binary-x86_64/share/cgatools-X.X.X/cgatools-
   X.X.X-docs/* /Users/complete/cgatools_docs
   ```

7. Make sure the new commands are available to the shell.

   ```
   4$ hash –r
   ```

8. Test the install:

   ```
   5$ cgatools
   ```

   If CGA Tools returns with the version number (for example "`1.5.0.0`") and a page of help notes, you have successfully installed CGA Tools.

At this point you can skip ahead to "Obtaining a Reference Human Genome for Use with CGA Tools."

Linux: Installing CGA Tools from a Binary Distribution

1. Create a directory structure.

   We recommend the following directory structure to support the installation and use of CGA Tools. This structure assumes your username is "complete" and all Complete Genomics data and applications are installed by user "complete".

Content	Directory Structure
Tar files and other intermediate files that are not needed after CGA Tools is installed	`/home/complete/src`
CGA Tools executable	`/home/complete/local/bin`
CGA Tools documentation	`/home/complete/local/share/cgatools-X.X.X/doc`
Your Complete Genomics human genome data files	`/home/complete/data`
Reference human genome sequence files	`/home/complete/data/ref`

The installation instructions assume this directory structure. If you choose to create a variation on this structure, make sure to adapt the installation instructions below to your structure.

2. Be sure that the directory containing the CGA Tools executable is in your $PATH.

   Consult your shell documentation for how to set the $PATH in your `.tcshrc` or `.bashrc` file.

3. The current CGA Tools binary distribution is available from SourceForge (http://cgatools.sourceforge.net). The files are named as follows, where the X's are replaced by the current version number:

   `cgatools-X.X.X.X-linux_binary-x86_64.tar.gz`

   Download the CGA Tools binary distribution into:

   `/home/complete/src`

4. Untar the tarball:

   ```
   1$ tar -xzf cgatools-X.X.X.X-linux_binary-x86_64.tar.gz
   ```

   This command creates a directory including CGA Tools executable and documents:

   - Executable: `/bin/cgatools`
   - Documentation: `/share/cgatools-X.X.X/cgatools-X.X.X-docs/index.html`

5. Copy the executable into your binary directory:

   ```
   2$ cp cgatools-X.X.X.X-linux_binary-x86_64/bin/cgatools /home/complete/bin
   ```

   If you are installing CGA Tools into a Linux system directory which requires sudo (that is, root privileges) you will need to enter:

   ```
   2$ sudo cp cgatools-X.X.X.X-linux-x86_64/bin/cgatools /usr/local/bin
   ```

6. Copy the documentation into your documentation directory:

   ```
   3$ cp cgatools-X.X.X.X-linux_binary-x86_64/share/cgatools-X.X.X/cgatools-X.X.X-
   docs/* /home/complete/local/share/cgatools-X.X.X/doc
   ```

7. Make sure the new commands are available to the shell.

   For C-shell (`csh, tcsh`):

   ```
   4$ rehash
   ```

   For Bash or Bourne shell:

   ```
   4$ hash –r
   ```

8. Test the install:

   ```
   5$ cgatools
   ```

   If CGA Tools returns with the version number (for example "`1.5.0.0`") and a page of help notes, you have successfully installed CGA Tools.

At this point you can skip ahead to "Obtaining a Reference Human Genome for Use with CGA Tools".

Installing CGA Tools in Galaxy

CGA Tools is packaged as repositories for installation within a Galaxy instance running on a local computer or a dedicated server. These repositories contain instructions for automated and manual installation of the tools and instructions for the download and installation of reference genome files.

You can use Galaxy's automatic installation to install the CGA Tools repository into your Galaxy instance:

- Log into Galaxy using an account with administrator privileges.
- Follow the instructions provided in the CGA Tools repository or in the Galaxy wiki topic "Installing Galaxy tool shed repository tools into a local Galaxy instance".

The CGA Tools repositories available in the Galaxy main tool shed are as follows:

- Linux installations of Galaxy: cg_cgatools_linux
- Max OS X installations of Galaxy: cg_cgatools_mac_osx

After you have installed the CGA Tools repository, you are ready to download a reference human genome and additional auxiliary files for use with CGA Tools. Follow the instructions provided in the CGA Tools repository.

> The CGA Tools distribution for Galaxy includes core tools such as listvariants, testvariants, calldiff, snpdiff, junctiondiff, join, and varfilter. Updates to the repositories are published independently of CGA Tools updates and include the latest version of CGA Tools available at the time of the repository update.

Installing CGA Tools from Source Code

CGA Tools is available as source code that you build in your own Linux environment.

> It is recommended that Mac OS X users install from the binary. If you find it is necessary to install CGA Tools from the source, please email support@completegenomics.com for assistance.

Preparing the Environment

1. Create a directory structure.

   We recommend the following directory structure to support the installation, build, and use of CGA Tools. This structure assumes your username is "complete" and all Complete Genomics data and applications are installed by user "complete".

Content	Directory Structure
Tar files and other intermediate files that are not needed after CGA Tools is installed	`/home/complete/src`
CGA Tools executable (and commands used by CGA Tools build process, including CMake)	`/home/complete/local/bin`
CGA Tools documentation	`/home/complete/local/share/cgatools-X.X.X/doc`
Your Complete Genomics human genome data files	`/home/complete/data`
Reference human genome sequence files	`/home/complete/data/ref`

The installation instructions assume this directory structure. If you choose to create a variation on this structure, make sure to adapt the installation instructions below to your structure.

2. Be sure that the directory containing the CGA Tools executable is in your $PATH.

   Consult your shell documentation for how to set the $PATH in your `.tcshrc` or `.bashrc` file.

Installing CMake

If you already have a correct version of CMake installed on your system, make sure the command is in your $PATH and skip to the next section "Installing Boost."

Note: Precompiled binaries are available for CMake. If you choose to use a CMake binary as opposed to build from source, make sure its machine architecture is the same as used for Boost. For example, 64-bit CMake should only be used with 64-bit Boost.

To build and install CMake from source:

1. Open a Linux/Unix command shell.
2. Download the CMake distribution into `/home/complete/src`

   http://www.cmake.org/cmake/resources/software.html
3. Change to the download directory.

   ```
   1$ cd /home/complete/src
   ```
4. Unpack the tarball:

   ```
   2$ tar -xvf cmake-2.8.1.tar.gz
   ```

   This creates a `cmake-2.8.1` subdirectory.
5. Change to the CMake directory.

   ```
   3$ cd cmake-2.8.1
   ```
6. Configure the software, specifying the final installation target:

   ```
   4$ ./bootstrap --prefix=/home/complete/local
   ```

   Resolve any errors, such as not having GNU C Compiler Collection installed. See "Software Requirements."
7. Build the software.

   ```
   5$ gmake
   ```

   This takes a few minutes. Resolve any errors.
8. Install the software.

   ```
   6$ make install
   ```

   Alternatively, if you are installing CMake into a system directory which requires root privileges (`sudo`) you will need to type:

   ```
   6$ sudo make install
   ```
9. Make sure the new commands are available to the shell.

   For C-shell (`csh, tcsh`):

   ```
   7$ rehash
   ```

   For Bash or Bourne shell:

   ```
   7$ hash -r
   ```

10. Test the installation.

    ```
    8$ cmake –-help
    ```

    If the `cmake` command prints the correct version number and a page of help notes, you have successfully installed CMake and your path variable is set correctly.

Installing Boost

To build and install Boost from source:

1. Download the Boost distribution into `/home/complete/src`.

   http://sourceforge.net/projects/boost/files/boost

2. Change to the download directory.

   ```
   9$ cd /home/complete/src
   ```

3. Unpack the tarball:

   ```
   10$ tar -xvf boost_1_46_1.tgz
   ```

   This will take several minutes. This creates a `boost_1_46_1` subdirectory.

4. Change to the Boost directory:

   ```
   11$ cd boost_1_46_1
   ```

5. Configure the software, specifying the final installation target:

   ```
   12$ ./bootstrap.sh --prefix=/home/complete/local
   ```

6. Now build the software:

   ```
   13$ ./bjam install
   ```

   Alternatively, if you are installing CMake into a system directory which requires sudo (that is, root privileges) you will need to type:

   ```
   13$ sudo ./bjam install
   ```

   This step will take a long time (perhaps an hour). You will see many `libboost*` files appear in `/home/complete/local/lib` and many `*.hpp` files in `/home/complete/local/include`.

   It is not uncommon to see some errors in the Boost install.

Downloading and Recompiling the CGA Tools Source Code

To build and install CGA Tools:

1. Download the CGA Tools distribution into `/home/complete/src`.

   The CGA Tools distribution has a name such as `cgatools-X.X.X.X-source.tar.gz` where the X's are replaced by the current version number. It is available here:

   http://sourceforge.net/projects/cgatools/files/

2. Change to the download directory.

   ```
   14$ cd /home/complete/src
   ```

3. Unpack the tarball:

   ```
   15$ tar -xvf cgatools-X.X.X.X-source.tar.gz
   ```

4. Change to the `cgatools-X.X.X.X-source` directory:

   ```
   16$ cd cgatools-X.X.X.X-source
   ```

5. Create an empty directory in which to build CGA Tools (`build` for example).

   ```
   17$ mkdir build
   ```

6. Change to the `build` directory:

   ```
   18$ cd build
   ```

7. Configure CGA Tools.

   You can type this next command on one line or use \ (backslash) to split it onto multiple lines, as shown here.

   ```
   19$ cmake -DBOOST_ROOT=/home/complete/local \
   -DCMAKE_INSTALL_PREFIX=/home/complete/local/ \
   -DCMAKE_BUILD_TYPE=Release \
   /home/complete/src/cgatools-X.X.X.X-source
   ```

   where the `DBOOST_ROOT` flag points to the final installation target for Boost.

8. Compile CGA Tools.

   ```
   20$ make -j8
   ```

9. Tests CGA Tools.

   This step uses Python to run the 75 CGA Tools test cases.

   ```
   21$ ctest -j8
   ```

   If any of the tests fail, investigate the failure before proceeding to the next step.

10. Install CGA Tools.

    ```
    22$ make -j8 install
    ```

    Alternatively, if you are installing CGA Tools into a system directory which requires root privileges (`sudo`) you will need to type:

    ```
    22$ sudo make -j8 install
    ```

11. Make sure the new commands are available to the shell.

    For C-shell (`csh, tcsh`):

    ```
    23$ rehash
    ```

    For Bash or Bourne shell:

    ```
    23$ hash -r
    ```

12. Test the install:

    ```
    24$ cgatools
    ```

    If CGA Tools returns with the version number (for example "`1.1.0.0`") and a page of help notes, you have successfully installed CGA Tools.

    If you are ultimately unable to install CGA Tools and are warned that there are problems with the Boost libraries, you may need to consult a system administrator for help.

13. Locate the documentation for CGA Tools:

    `/home/complete/local/share/cgatools-X.X.X/doc/index.html`

Obtaining a Reference Human Genome for Use with CGA Tools

Complete Genomics supports two references. The first, which we refer to as "build 36," consists of the assembled nuclear chromosomes from NCBI build 36 (not unplaced or alternate loci) plus Yoruban mitochondrion NC_001807.4. This assembly is also known as UCSC hg18. The second reference, which we refer to as "build 37," consists of the assembled nuclear chromosomes from GRCh37 (not unplaced or alternate loci), plus the Cambridge Reference Sequence for the mitochondrion (NC_012920.1). This assembly (though with an alternate mitochondrial sequence) is also known as UCSC hg19.

CGA Tools uses a compact representation of the human genome in a specialized CRR (Compact Randomly accessible Reference) format. Customers select either build36 or build37 as the reference assembly for the samples they submit to Complete Genomics, and must use the appropriate CRR file (either ***build36.crr*** or ***build37.crr***) when analyzing their genome data using CGA Tools.

There are two ways to obtain the CRR file associated with your build.

1. We recommend that you download the relevant file from the Complete Genomics FTP site.
2. Alternatively, you can Building the CRR File from FASTA Files that you download from the Complete Genomics FTP site.

In each case, you will want to verify the content of the file, as described in "Verifying CRR File Content."

Downloading the CRR File

1. Download the CRR files into:

   ```
   Linux:        /home/complete/data/ref
   Mac OS X:     /Users/complete/data/ref
   ```

   The CRR files are available here:

   ftp://ftp.completegenomics.com/ReferenceFiles/build36.crr

   ftp://ftp.completegenomics.com/ReferenceFiles/build37.crr

Next, verify the CRR file content, as described in "Verifying CRR File Content."

Building the CRR File from FASTA Files

Important: Complete Genomics "build37" consists of the assembled nuclear chromosomes from GRCh37 (not unplaced or alternate loci), plus the Cambridge Reference Sequence for the mitochondrion (NC_012920.1). This assembly (though with an alternate mitochondrial sequence) is also known as UCSC hg19. Customers who build the CRR files as described below must use the correct mitochondrial sequence. CRR files generated using the UCSC hg19 FASTA files are incompatible with CGA Tools because they contain a different mitochondrial sequence. Compatible FASTA files are available from Complete Genomics, as described below.

Generate the CRR files using the CGA Tools fasta2ccr command:

1. Download the FASTA files into:

   ```
   Linux:        /home/complete/src

   Mac OS X:     /Users/complete/data/ref/temp
   ```

   The FASTA sequences are available here:

   ftp://ftp.completegenomics.com/ReferenceFiles/build36.fa.bz2

   ftp://ftp.completegenomics.com/ReferenceFiles/build37.fa.bz2

2. Change to the download directory.

   ```
   Linux $       $ cd /home/complete/src
   Mac OS X $    $ cd /Users/complete/data/ref/temp
   ```

3. Perform the format conversion, making the CRR file. For example:

```
$ cgatools fasta2crr --input build36.fa.bz2 --output build36.crr
```

Next, verify the CCR file content, as described in "Verifying CRR File Content."

Verifying CRR File Content

1. List the contents of the CRR file using the CGA Tools listcrr command. For example:

```
$ cgatools listcrr --reference build36.crr
```

For build 36 CRR files, the output should be identical to the following:

```
ChromosomeId Chromosome    Length Circular Md5
           0       chr1 247249719    false 9ebc6df9496613f373e73396d5b3b6b6
           1       chr2 242951149    false b12c7373e3882120332983be99aeb18d
           2       chr3 199501827    false 0e48ed7f305877f66e6fd4addbae2b9a
           3       chr4 191273063    false cf37020337904229dca8401907b626c2
           4       chr5 180857866    false 031c851664e31b2c17337fd6f9004858
           5       chr6 170899992    false bfe8005c536131276d448ead33f1b583
           6       chr7 158821424    false 74239c5ceee3b28f0038123d958114cb
           7       chr8 146274826    false 1eb00fe1ce26ce6701d2cd75c35b5ccb
           8       chr9 140273252    false ea244473e525dde0393d353ef94f974b
           9      chr10 135374737    false 4ca41bf2d7d33578d2cd7ee9411e1533
          10      chr11 134452384    false 425ba5eb6c95b60bafbf2874493a56c3
          11      chr12 132349534    false d17d70060c56b4578fa570117bf19716
          12      chr13 114142980    false c4f3084a20380a373bbbdb9ae30da587
          13      chr14 106368585    false c1ff5d44683831e9c7c1db23f93fbb45
          14      chr15 100338915    false 5cd9622c459fe0a276b27f6ac06116d8
          15      chr16  88827254    false 3e81884229e8dc6b7f258169ec8da246
          16      chr17  78774742    false 2a5c95ed99c5298bb107f313c7044588
          17      chr18  76117153    false 3d11df432bcdc1407835d5ef2ce62634
          18      chr19  63811651    false 2f1a59077cfad51df907ac25723bff28
          19      chr20  62435964    false f126cdf8a6e0c7f379d618ff66beb2da
          20      chr21  46944323    false f1b74b7f9f4cdbaeb6832ee86cb426c6
          21      chr22  49691432    false 2041e6a0c914b48dd537922cca63acb8
          22       chrX 154913754    false d7e626c80ad172a4d7c95aadb94d9040
          23       chrY  57772954    false 62f69d0e82a12af74bad85e2e4a8bd91
          24       chrM     16571     true d2ed829b8a1628d16cbeee88e88e39eb
```

For build 37 CRR files, the output should be identical to the following:

```
ChromosomeId Chromosome    Length Circular Md5
           0       chr1 249250621    false 1b22b98cdeb4a9304cb5d48026a85128
           1       chr2 243199373    false a0d9851da00400dec1098a9255ac712e
           2       chr3 198022430    false 641e4338fa8d52a5b781bd2a2c08d3c3
           3       chr4 191154276    false 23dccd106897542ad87d2765d28a19a1
           4       chr5 180915260    false 0740173db9ffd264d728f32784845cd7
           5       chr6 171115067    false 1d3a93a248d92a729ee764823acbbc6b
           6       chr7 159138663    false 618366e953d6aaad97dbe4777c29375e
           7       chr8 146364022    false 96f514a9929e410c6651697bded59aec
           8       chr9 141213431    false 3e273117f15e0a400f01055d9f393768
           9      chr10 135534747    false 988c28e000e84c26d552359af1ea2e1d
          10      chr11 135006516    false 98c59049a2df285c76ffb1c6db8f8b96
          11      chr12 133851895    false 51851ac0e1a115847ad36449b0015864
          12      chr13 115169878    false 283f8d7892baa81b510a015719ca7b0b
          13      chr14 107349540    false 98f3cae32b2a2e9524bc19813927542e
          14      chr15 102531392    false e5645a794a8238215b2cd77acb95a078
          15      chr16  90354753    false fc9b1a7b42b97a864f56b348b06095e6
          16      chr17  81195210    false 351f64d4f4f9ddd45b35336ad97aa6de
          17      chr18  78077248    false b15d4b2d29dde9d3e4f93d1d0f2cbc9c
          18      chr19  59128983    false 1aacd71f30db8e561810913e0b72636d
          19      chr20  63025520    false 0dec9660ec1efaaf33281c0d5ea2560f
          20      chr21  48129895    false 2979a6085bfe28e3ad6f552f361ed74d
          21      chr22  51304566    false a718acaa6135fdca8357d5bfe94211dd
          22       chrX 155270560    false 7e0e2e580297b7764e31dbc80c2540dd
          23       chrY  59373566    false 1e86411d73e6f00a10590f976be01623
          24       chrM     16569     true c68f52674c9fb33aef52dcf399755519
```

2. Move the verified reference to a final location.

```
Linux        $ mv build36.crr /home/complete/data/ref
Mac OS X     $ mv build36.crr /Users/complete/data/ref
```

3. Use the CGA Tools decodecrr command to test your reference file by extracting a sequence from the reference genome based on user-defined coordinates. For example:

```
$ cgatools decodecrr \
--reference /home/complete/data/ref/build36.crr \
--range chr16:10000000-10000050
```

Obtaining Ancillary Files for Use with CGA Tools

The following files are required to generate certain types of CGA Tools output, and can be downloaded from the Complete Genomics FTP server.

- RepeatMasker (from UCSC Genome Browser track) — Database of DNA sequences for interspersed repeats and low complexity DNA sequences.
  - Build 36 — ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/rmsk36.tsv.gz
  - GRCh37 — ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/rmsk37.tsv.gz
- Segmental Duplication (from UCSC Genome Browser track) — Duplications of more than 1000 bases of non-RepeatMasked sequence. [genome.ucsc.edu/cgi-bin/hgTrackUi?hgsid=214405809&g=genomicSuperDups]
  - Build 36 — ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/segdup36.tsv.gz
  - GRCh37 — ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/segdup37.tsv.gz

- Human Genome Reference Assembly Gene Annotation files — NCBI reference assembly and alignment data.
  - Build 36 — ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/gene_36_NCBI36.3.tsv.gz
  - GRCh37
    - Annotation build 37.2 (Analysis Pipelines 2.4 and earlier): ftp://ftp.completegenomics.com/AnnotationFiles/version2.0.0/gene_37_NCBI37.2.tsv.gz
    - Annotation build 104 (Analysis Pipeline 2.5): ftp://ftp.completegenomics.com/AnnotationFiles/version2.5.0/gene_37_NCBI104.tsv.gz
- FASTA sequence — Human Genome Reference Sequence files converted to FASTA format:
  - Build 36 — ftp://ftp.completegenomics.com/ReferenceFiles/build36.fa.bz2
  - GRCh37 — ftp://ftp.completegenomics.com/ReferenceFiles/build37.fa.bz2
- Variant Score Calibration Data — Data based on replicate experiments conducted by Complete Genomics at various levels of coverage. These files are required to run CGA Tools mkvcf (when producing calibrated scores) and calldiff (when producing SomaticOutput) tools. Note that samples analyzed using Analysis Pipeline 2.5 require version 3 of the calibration data, available here:
  - ftp://ftp.completegenomics.com/ScoreCalibrationFiles/var-calibration-v3.tgz

  After downloading the calibration data, you must untar it using the following command:

  ```
  tar -xzf /path/to/var-calibration-v3.tgz
  ```

Congratulations. You are now ready to use CGA Tools!

Genome Comparison Tools

A Note on Conventions

To call low certainty regions or "no-call" regions, Complete Genomics augments the alphabet {A, C, G, T} with two additional characters:

The "N" character corresponds to a one-base sequence that may be any base (A, C, G, or T).

The "?" character corresponds to zero or more bases of unknown sequence.

The Problem of Genome Comparison

Genome comparison is the process where genomic sequences are compared to determine whether they are identical or different from each other. Comparing genomic sequence containing no-called bases is more challenging because of the additional uncertainty of the sequence. In this case, it may be possible to demonstrate that two sequences differ, or that they are compatible with each other, but it may not be possible to demonstrate they are identical. With genome comparisons, there are three common tasks:

1. Is a genome identical, different, or compatible with the reference genome at a given location?
2. Is a genome identical, different, or compatible with a known common sequence?
3. Is a genome identical, different, or compatible with another genome at a given location?

The particular way a genome is described by re-sequencing technologies goes a long way towards solving genome comparison problems 1 and 2: genomes are represented as a set of differences (or variants) against the reference genome. The Complete Genomics variant file format differs from most other common variant file formats in that in addition to describing the variants, it also distinguishes regions of the genome that are called as reference from those that are no-called. As we will see later, this distinction is essential in solving many comparison problems.

The following example shows how the Complete Genomics ***variations*** file describes the situation where chr1 is a diploid chromosome and chr2 is a haploid chromosome:

```
chr1 reference:     CATGACCCGCAAA-TCTGAAACTATCTGGCCCTTGGCAGGGG--A
chr1 haplotype 1:   ?ATGACCTGCAAAATCTGAAACT--CTGGCCCTTGGCAGGGGGGA
chr1 haplotype 2:   ?ATGACCCGCAAAATCTGAAACTATCTGGCTNTTGGCAGGGT--A

chr2 reference:     TGATATTTTTCATCAACATTACAGGCA
chr2:               TGATATTTTTNATCAACACGACAGGCA
```

Figure 3 shows the corresponding variant file.

Figure 3: Variant File

>locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink	xRef	alleleFreq	alternativeCalls
1	2	all	chr1	0	1	no-call	=	?							
2	2	all	chr1	1	7	ref	=	=							
3	2	1	chr1	7	8	snp	C	T	87	87	PASS	1	dbsnp:123		
3	2	2	chr1	7	8	ref	C	C	57	57	PASS	2	dbsnp:123		
4	2	all	chr1	8	13	ref	=	=							
5	2	1	chr1	13	13	ins		A	15	36	VQLOW				
5	2	2	chr1	13	13	ins		A	19	42	VQLOW				
6	2	all	chr1	13	22	ref	=	=							
7	2	1	chr1	22	24	del	AT		60	47	PASS	1			
7	2	2	chr1	22	24	ref	AT	AT	75	55	PASS	2			
8	2	all	chr1	24	29	ref	=	=							
9	2	1	chr1	29	31	ref	CC	CC	57	57	PASS	1			
9	2	2	chr1	29	31	no-call-ri	CC	TN	65	65	PASS	2			
10	2	all	chr1	31	40	ref	=	=							
11	2	1	chr1	40	41	ref	G	G	129	101	PASS	1			
11	2	1	chr1	41	41	ins		GG	118	120	PASS	1			
11	2	2	chr1	40	41	snp	G	T	479	479	PASS	2			
12	2	all	chr1	41	42	ref	=	=							
13	1	all	chr2	0	10	ref	=	=							
14	1	1	chr2	10	11	no-call-rc	C	N	50	47	PASS				
15	1	all	chr2	11	18	ref	=	=							
16	1	1	chr2	18	20	sub	TT	CG	102	102	PASS				
17	1	all	chr2	20	27	ref	=	=							

The genome is first aligned to the reference, and then split into loci. Each locus may describe multiple alleles (if ploidy > 1), and for each allele at each locus, there may be one or more lines (or "calls") to describe the sequence. The variant file describes 0-based offsets within the reference chromosome.

In the variant file in Figure 3, locus 3 describes a heterozygous SNP (one-base polymorphism on one allele, reference on the other allele). Locus 5 describes a homozygous insertion. The allele column is used to distinguish the alleles of calls within a locus. For example, the "ref" and "ins" calls of locus 11 are on the same allele, whereas the "snp" call is on the opposite allele. To show that two calls of different loci are on the same haplotype, the format uses the *hapLink* field. Calls known to be on the same haplotype have the same *hapLink* value; calls with different *hapLink* values may or may not be on the same haplotype (the phasing is not known).

For a detailed reference of the Complete Genomics variant (***var***) file format, see the Data File Formats document.

Problems Not Solved by Variant File Format

One challenge of genome comparison is that aligning a genome to the reference is not always consistent. For example in Figure 3, the homozygous insertion at locus 5 could have also been described by the same homozygous insertion three bases to the left. Or the substitution at locus 16 could have been described as two SNPs. Comparing two genomes that describe the same sequence in different ways can be complicated.

We could make canonicalization rules such as "always use the leftmost insertion for any insertion that has multiple possible representations" or "always decompose an allele consisting of a SNP, two reference bases, then another SNP, into separate calls." Indeed, Complete Genomics has rules like these that are

generally followed. But there are at least three remaining problems in solving the genome comparison problems described above:

- Known variants are not always described in their canonical form.

  For example, entries rs34330821 and rs34544546 in the dbSNP database of known variants describe equivalent insertions that are 18 bases apart. This may seem superficial, in that dbSNP entries that are not described in their canonical form can be canonicalized. But if our canonical form uses less decomposition than the dbSNP submission, this may not be possible; if a dbSNP submission has been decomposed, the submission has lost information about nearby variants that exist on the same haplotype.

- Canonical forms of near-identical sequences may be more different than the non-canonical forms.

  For example, suppose we have a genome that is equivalent to a SNP and an insert against the reference, as described in canonicalization 1:

  ```
  Reference:          TG A TGTGAATTGGTG ---------------------------- AGT
  Canonicalization 1: TG C TGTGAATTGGTG TAGTGTGAATGAGTGTGTGAATTGGTG AGT

  Reference:          TG A---------------------------- TGTGAATTGGTGAGT
  Canonicalization 2: TG CTGTGAATTGGTGTAGTGTGAATGAGTG TGTGAATTGGTGAGT
  ```

  The insert in canonicalization 1 might be the simplest way to describe the genome if the SNP did not exist. But one could argue that the single substitution in canonicalization 2 is the simplest canonicalization of the genome, given that the SNP does exist and the TGTGAATTGGTG sequence is repeated immediately after the SNP, in the inserted region. (This would be the case for a canonicalization which favors fewer calls.) It is not obvious by visual inspection that the insert from canonicalization 1 and the substitution of canonicalization 2 differ by only a SNP.

- No-calls may not be canonicalized like insertions or deletions, such that an insert may be compatible with another genome only when viewing a larger sequence of the genome.

  For example, suppose we have the following reference and the following genome:

  ```
  Reference:  CGAAAAAAA-TTTTCG
  Genome:     CGAAAAAAAATTTTCG
  ```

  Now suppose the genome reconstruction process discovers that an insertion has occurred, but it does not know if the first base in the run of A's is really an A, or perhaps was a C. In this case, we are forced to align the no-call at the beginning as follows:

  ```
  Reference:  CG-AAAAAAATTTTCG
  Genome:     CGNAAAAAAATTTTCG
  ```

  Length no-calls ("?") may further complicate the situation so that the alignment is unclear. For example, suppose in the same example above, in addition to not knowing if the first base of the run is an A or a C, we also don't know the length of the run of A's at all. Suppose also that we know that the run of T's has increased in length from four to five. There could be at least two reasonable alignments of the result, corresponding to a called insert or a called SNP:

  ```
  Reference:   CGAAAAAAATTTT-CG
  Alignment 1: CG?AAAAAATTTTTCG
  Reference:   CG-AAAAAAATTTTCG
  Alignment 2: CG?AAAAAATTTTTCG
  ```

Genome Comparison with CGA Tools

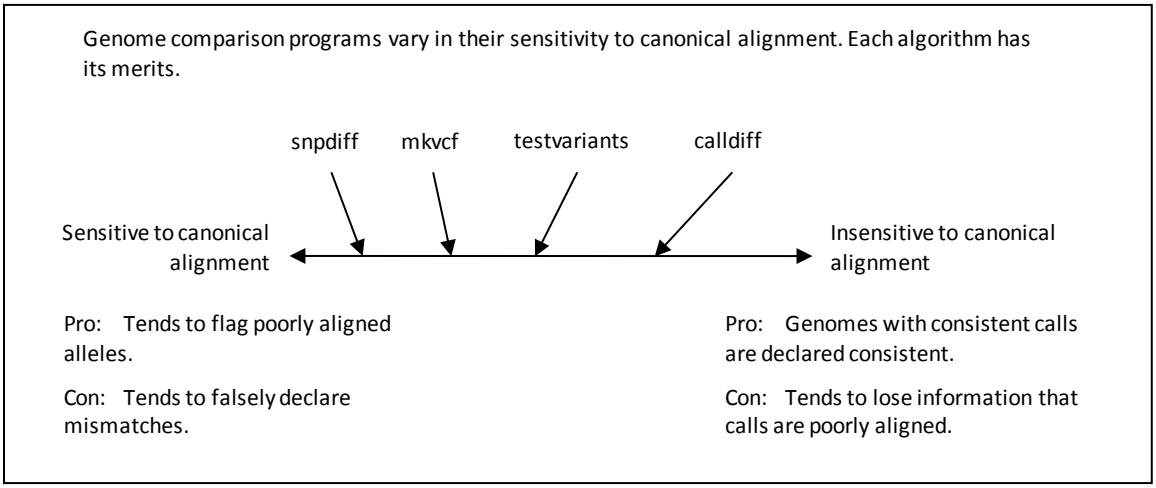
There is a wide spectrum of useful genome comparison methods, which range in their sensitivity to the canonical alignment of called sequence. Algorithms that are very sensitive to canonical alignment tend to declare sequences inconsistent when in fact they are consistent. Algorithms that are less sensitive to canonical alignment tend to be less discriminating in terms of the quality of the alignment of called sequence.

CGA Tools includes four small variant genome comparison utilities that provide varying degrees of sensitivity to inconsistent canonical alignments:

- snpdiff can be used to compare the results of a SNP caller to a Complete Genomics variant file. It is quite sensitive to the canonical alignment of called sequence.
- calldiff can be used to compare two variant files. Because calldiff has access to all the aligned calls from the two genomes being compared, it is capable of performing sequence comparison to determine whether calls in the two genomes with different alignment are in fact, the same call. This capability makes calldiff less sensitive to canonical alignment of called sequence than other CGA Tools that perform genome comparison.
- listvariants/testvariants can be used to compare multiple variant files. Because testvariants takes an individual variant and compares it to all the calls in a superlocus from each genome, it only has access to the aligned calls from one genome and not to what variants are expected to be nearby the individual variant it is testing for. Thus, the sensitivity to canonical alignment of called sequence will be somewhere between snpdiff and calldiff.
- mkvcf can be used to compare multiple variant files and has sensitivity to canonical alignment between snpdiff and testvariants. mkvcf differs from listvariants/testvariants in that it provides output in VCF 4.1 format and contains the rich set of annotations Complete Genomics provides with each dataset.

Figure 4 illustrates the tradeoffs among the utilities.

Figure 4: Sensitivity of Genome Comparison Algorithms



snpdiff

Compares genotype calls to Complete Genomics ***var*** or ***masterVarBeta*** files. The following sections describe its behavior:

Synopsis

```
cgatools snpdiff --help
       --genotypes <genotype_file>
       --variants <variants_file>#<filter1>,<filter2>
       --reference <crr_file>
       --output-prefix <prefix>
       --reports <report_type>[,<report_type>]
```

Description

The snpdiff tool compares genotype calls to Complete Genomics ***var*** and ***masterVarBeta*** files. It is particularly useful for comparing a Complete Genomics variant file to genotype calls provided by an alternative sequencing or genotyping platform.

Analysis Pipeline Version Effects

- Analysis Pipeline version 2.4 introduced updated ***var*** and ***masterVarBeta*** file formats to provide new features such as ambiguous calling, minor allele frequencies from the 1000 Genomes Project, and changes to the variant quality flagging system. When operating on files generated before Analysis Pipeline 2.4, snpdiff converts the output to the version 2.4 file format. It provides variant quality flags in the *varFilter* column (instead of *varQuality*) and adds *alleleFreq* and *alternativeCalls* columns whose fields are left empty.
- Analysis Pipeline version 2.0 introduced two independent scoring methods to quantify confidence in the reported call for each allele: Variable Allele Fraction (VAF) Scoring and Equal Allele Fraction (EAF) Scoring. The ***var*** and ***masterVar*** files generated after 2.0 include the *varScoreVAF* and *varScoreEAF* fields. When operating on files generated before 2.0, snpdiff fills *varScoreVAF* and *varScoreEAF* fields with the previously used metric *totalScore.* The *varFilter* field is left empty.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--genotypes <genotype_file>`	The input genotypes file.
`--variants <variants_file>#<filter1>,<filter2>`	The input variant file, a ***var*** or ***masterVarBeta*** file. Optionally you can include filters used to turn selected calls into no-calls. See the filtering syntax in the varfilter (beta) tool description.
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file.

Option	Description
`--output-prefix <prefix>`	The path prefix for all output reports. This prefix can be used in two ways: ▪ If a path is specified (for example "`/home/myFiles`"), report files are saved to that location. ▪ If a string is specified (for example "`Run20111011`") it is appended to the start of the filename, and the file will be saved in the active directory.
`--reports <report>[,<report>...]`	Comma-separated list of reports to generate. A report is one of: ▪ Output: Standard output report. ▪ Verbose: Verbose report. ▪ Stats: Statistics report.

Input Files

snpdiff takes as input a genotype calls file, a Complete Genomics variant (***var-[ASM-ID].tsv.bz2***) or masterVar file (***masterVarBeta-[ASM-ID].tsv.bz2***), and a reference CRR file of the appropriate build. In addition, you can filter the input file to turn selected calls into no-calls, as described for varfilter (beta).

The input genotype calls file must be a tab-delimited file with columns indicating the data to be compared. Table 1 lists the possible columns; note that the order of columns is not significant, but column titles must be conserved. Any additional columns are passed directly to the output. Figure 5 shows an example input file.

Table 1: Input Columns used for Comparison

Column Name	Usage Notes
`SNPID`	(Optional)
`SNP`	(Optional)
`Chromosome`	(Required)
`Offset0Based`	(Required) The base positions must be represented in zero based coordinates.
`GenotypesStrand`	(Optional)
`Genotypes`	(Optional) snpdiff can be run without the *Genotypes* column, for example, to see what variants are present in both datasets. The column must be present to actually compare the genotype and genome calls.

Figure 5: SNP Calls As Input to snpdiff

SNPID	SNP	Chromosome	Offset0Based	GenotypesStrand	Genotypes
rs2775537	A/G	chr21	14601414	+	AA
rs2742158	C/T	chr21	14638915	+	CC
rs2792379	C/T	chr21	15105484	+	TT
rs2822127	A/G	chr21	15170634	+	GG
rs1985740	C/T	chr21	15173845	+	CC
rs3115511	A/G	chr21	15214707	+	AA
rs2822432	C/T	chr21	15516947	+	CC

Output File Format: Standard and Verbose Reports

Table 2: Columns in snpdiff Output and Verbose Reports

Output	Verbose	Column	Description
1	1	SNPID	SNP identifier from input genotypes file.
2	2	SNP	Possible alleles at this SNP.
3	3	Chromosome	Chromosome name of SNP in text: `chr1`, `chr2`,...,`chrX`, `chrY`, `chrM` from the input genotypes file.
4	4	Offset0Based	Coordinate of SNP in zero-based coordinates.
5	5	GenotypesStrand	(If present in the input file) The strand of the calls in the *Genotypes* column (+ or -, defaults to +).
6	6	Genotypes	(If present in the input file) The calls, one per allele from the genotypes file. The following calls are recognized: A,C,G,T A called base. N A no-call. - A deleted base. . A non-SNP variation.
7	7	Reference	The reference base at the given position.
8	8	VariantFile	The Complete Genomics variant file calls, one per allele. The character codes are the same as is described for the *Genotypes* column.
9		DiscordantAlleles	(Standard Output only) The number of alleles that are discordant with calls in the *VariantFile* column. If the *VariantFile* is haploid at the given position but two alleles are provided in the *Genotypes* column, each genotype allele is compared against the haploid call of the *VariantFile*.
10		NoCallAlleles	(Standard Output only) The number of alleles that were no-called in the *VariantFile* column. If the *VariantFile* is haploid at the given position but two alleles are provided in the *Genotypes* column, then a *VariantFile* no-call is counted twice.
	9	Locus	Identifier (index) of a locus.
	10	Ploidy	The ploidy of the reference genome at the locus (= 2 for autosomes, 2 for pseudo-autosomal regions on the sex chromosomes, 1 for males on the non-pseudo-autosomal parts of the sex chromosomes, 1 for mitochondrion, 2 if *varType* is `no-ref` or `PAR-called-in-X`). The reported ploidy is fully determined by gender, chromosome and location, and is not inferred from the sequence data.
	11	Allele	Identifier for each allele at the locus.
	12	chromosome	Chromosome name of SNP in text: chr1, chr2,...,chrX, chrY, chrM from Complete Genomics variant file.
	13	Begin	Coordinate specifying the base position of the SNP using half-open, 0 based coordinates.
	14	End	Coordinate specifying the base position of the SNP using half-open, 0 based coordinates.
	15	varType	Type of variation: one of `snp`, `ins`, `del`, `sub`, `ref`, `no-call-rc`, `no-call-ri`, `no-call`, `No-ref`, or `PAR-called-in-X`.
	16	Reference	Base call on the reference sequence.
	17	alleleSeq	Observed sequence of the allele in the Complete Genomics variation file.
	18	varScoreVAF	Variable allele fraction model confidence score from the Complete Genomics variation file (*varScoreVAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is populated with the *totalScore*. This field is empty for reference calls or no-calls.

Output	Verbose	Column	Description
	19	varScoreEAF	Equal allele fraction model confidence score from the Complete Genomics variation file (*varScoreEAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is equivalent to the *totalScore*. This field is empty for reference calls or no-calls.
	20	varFilter	List of indicators of low-quality or incomplete resolution of the variant call. If "PASS", then the allele passes all relevant quality tests. Otherwise, the list includes one or more semicolon-separated values from the following possible filters: ▪ `VQLOW` — indicates the call is homozygous and *allele1VarScoreVAF* is less than 20 dB, or the call is not homozygous and *allele1VarScoreVAF* is less than 40 dB. ▪ `SQLOW` — Indicates somatic variant has *somaticScore* < -10. ▪ `FET30` — Indicates somatic variant has *fisherSomatic* < 30. ▪ `AMBIGUOUS` — For homozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 10 dB of the call; for heterozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 20 dB of the call.
	21	hapLink	Identifier that links an allele at one locus to alleles at other loci. Currently only populated for very proximate variations that were either assembled together or were determined to be in phase using a correlation-based analysis between two variation intervals one mate pair away. Calls that share a hapLink identifier are expected to be on the same haplotype. Calls with haplinks appearing only once in the file and calls with no haplinks can be interpreted similarly: there is no phasing information with any other loci.
	22	xRef	Field containing external variation identifiers, populated for variations corroborated directly by dbSNP. Format for dbSNP: `dbsnp.<build>:<rsID>`, with multiple entries separated by the semicolon (;). <build> indicates in which build of dbSNP this entry first appeared. For example, "dbsnp.129:rs12345".

<table>
<tr><th>Output</th><th>Verbose</th><th>Column</th><th>Description</th></tr>
<tr><td></td><td>23</td><td>alleleFreq</td><td>Allele frequency value(s) for the entire call or for parts of the call that are corroborated directly by external sources. The source is 1000 Genomes Project minor allele frequency information in dbSNP.

Format is <code><source>:<frequency></code>, with multiple pairs separated by a semicolon. Precision of <code>frequency</code> is three decimal places.

Format for dbSNP becomes <code>dbsnp:<frequency></code>. Multiple entries for the same type of source mirror the multiple entries for this source appearing in xRef.

If an allele frequency value is not available for a dbSNP record, the corresponding position in the alleleFreq column is left empty.

<table>
<tr><th>When the call matches...</th><th>The alleleFreq field shows...</th></tr>
<tr><td>1 rsID with known frequency.</td><td><code>dbsnp:0.234</code></td></tr>
<tr><td>1 rsID with unknown frequency.</td><td>(empty string)</td></tr>
<tr><td>2 rsIDs (independently or as a combination) with both known frequencies.</td><td><code>dbsnp:0.234;dbsnp:0.123</code></td></tr>
<tr><td>2 rsIDs with both unknown frequencies.</td><td><code>;</code> (a single semicolon)</td></tr>
<tr><td>3 rsIDs with 1 known and 2 unknown frequencies.</td><td>Depending on which of the frequencies are unknown, one of the following:
<code>dbsnp:0.234;;</code>
<code>;dbsnp:0.234;</code>
<code>;;dbsnp:0.234</code></td></tr>
</table>
</td></tr>
<tr><td></td><td>24</td><td>alternativeCalls</td><td>Contains alternate calls for alleles designated “<code>AMBIGUOUS</code>”. Formatted as a semicolon-separated list of <code><sequence>:<score></code> pairs, where <code><sequence></code> is a hypothesized nucleotide sequence, and <code><score></code> is the score of that hypothesized sequence, relative to the called sequence.

For example, if alternativeCalls is “<code>AG:-1;G:-8</code>”, then sequence AG scored 1 dB less than the called sequence and G scored 8 dB less than the called sequence.</td></tr>
</table>

Figure 6 shows an example of the snpdiff Output file produced from the input shown in Figure 3.

Figure 6: snpdiff Output Report Example

SNPID	SNP	Chromosome	Offset0Based	GenotypesStrand	Genotypes	Reference	VariantFile	DiscordantAlleles	NoCallAlleles
rs2792379	C/T	chr21	15105484	+	TT	C	TT	0	0
rs219674	A/G	chr21	27728433	+	AA	A	NN	0	2
rs4817113	G/T	chr21	27776818	+	GG	G	AG	1	0
rs2164171	C/T	chr21	27789057	+	CC	T	CC	0	0
rs2830331	C/T	chr21	28034834	+	TT	T	.-	0	0
rs2830427	A/C	chr21	28085291	+	CC	A	CC	0	0

The result for each allele described in the *VariantFile* column above are any base call (“A”, “C”, “G”, or “T”), a no-call (“N”), a deletion (“-”), or a variation that is not consistent with a genotype at all (“.”). To compare the genotype calls to the calls in the variant file, snpdiff first determines the variant file calls at the given position. The algorithm used is sensitive to the canonical alignment, and it is aggressive in terms of making a base call at positions where the call does not have a *varType* of “snp” or “ref”. That being said, it is tested to be largely concordant with calls made by several alternative technologies. A discordance found by snpdiff is likely to be a true discrepancy between the calls made by the SNP caller and the variant file, rather than a mis-alignment of the two. For more information about the algorithm, see “snpdiff Algorithm” in the Appendix.

Figure 7 shows an example of the snpdiff Verbose Output file produced from the input shown in Figure 3.

Figure 7: snpdiff Verbose Report Output

SNPID	SNP	Chromosome	Offset0Based	GenotypesStrand	Genotypes	Reference	VariantFile	locus	ploidy	allele	chromosome
rs2792379	C/T	chr21	15105484	+	TT	C	T	21248606	2	1	chr21
rs2205585	C/T	chr21	17166670	+	CT	C	C	21267360	2	1	chr21
rs2823595	A/T	chr21	17442386	+	AA	T	A	21269698	2	1	chr21
rs4817113	G/T	chr21	27776818	+	GG	G	A	21375235	2	2	chr21
rs370092	A/G	chr21	46021606	+	GG	A	G	21511833	2	1	chr21

Figure 7: snpdiff Verbose Report Output (continued)

begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink	xRef	alleleFreq	alternativeCalls
15105484	15105485	snp	C	T	40	40	PASS		dbsnp.100:rs2792379		
17166670	17166671	ref	C	C	47	28	PASS				
17442386	17442387	snp	T	A	169	169	PASS		dbsnp.100:rs2823595		
27776818	27776819	snp	G	A	179	179	PASS	5332744			
46021606	46021607	snp	A	G	170	151	PASS	4604861	dbsnp.80:rs370092		

Output File Format: Stats Report

The Stats report is a comma-separated file with tables describing the results of the SNP comparison for each diploid genotype. There are three classes of tables that describe the comparison results (column headers) versus the genotype classifications (row labels):

- Locus classification: describes detailed match classifications (Figure 8 and Figure 9). These are contingency tables of genotype classifications by comparison results. Results are described both as a fraction of the total and by counts.
- Locus concordance: rolls match classifications into “discordance” and “no-call” (Table 10 and Table 11). A locus is considered discordant if it is discordant for either allele. A locus is considered no-call if it is concordant for both alleles but has a no-call on either allele. Results are described both as a fraction of the total and by counts.
- Allele concordance: similar to locus concordance but describes the comparison results for each allele separately (Table 12 and Table 13).

Table 3 and Table 4 describe the genotype and comparison classification values used in the Stats reports.

Table 3: Genotype Classifications

Class	Description
ref-ref	Both alleles from the genotypes file match the Complete Genomics variations file, which matches reference.
het-ref-alt	The alleles are heterozygous with one reference allele and one alternative allele.
het-alt-alt	The alleles are heterozygous with both alleles being non-reference.
hom-alt-alt	The alleles are homozygous and both are non-reference.

Table 4: Comparison Classifications

Class	Description
match-match	Both alleles from the genotypes file match the Complete Genomics variations file.
nocall-match	One allele is a nocall and one allele matches the Complete Genomics variations file.
nocall-nocall	Both alleles are not called.
match-mismatch	One allele from the genotypes file matches the Complete Genomics variations file while the other one does not.
mismatch-mismatch	Both alleles from the genotypes file disagree with the Complete Genomics variations file.
nocall-mismatch	One allele is a nocall and the other allele does not match the Complete Genomics variations file.

Class	Description
match-nonsnp	One allele matches while the other variant is not a SNP.
nonsnp-nonsnp	Neither allele is a SNP.
nocall-nonsnp	One allele is a nocall and the other is not a SNP.

The following figures show examples of the three table types.

Figure 8: Locus Classification by Fraction of Total

Locus classification by fraction of total

Genotype	match-match	nocall-match	nocall-nocall	match-mismatch	mismatch-mismatch	nocall-mismatch	match-nonsnp	mismatch-nonsnp	nonsnp-nonsnp	nocall-nonsnp	total
ref-ref	0.9814	0.00465	0.01163	0.00233	0	0	0	0	0	0	1
het-ref-alt	0.94395	0.0295	0.02655	0	0	0	0	0	0	0	1
het-alt-alt	0	0	0	0	0	0	0	0	0	0	0
hom-alt-alt	0.96727	0.02909	0.00364	0	0	0	0	0	0	0	1
total	0.96552	0.01916	0.01437	0.00096	0	0	0	0	0	0	1

Figure 9: Locus Classification by Count

Locus classification by count

Genotype	match-match	nocall-match	nocall-nocall	match-mismatch	mismatch-mismatch	nocall-mismatch	match-nonsnp	mismatch-nonsnp	nonsnp-nonsnp	nocall-nonsnp	total
ref-ref	422	2	5	1	0	0	0	0	0	0	430
het-ref-alt	320	10	9	0	0	0	0	0	0	0	339
het-alt-alt	0	0	0	0	0	0	0	0	0	0	0
hom-alt-alt	266	8	1	0	0	0	0	0	0	0	275
total	1008	20	15	1	0	0	0	0	0	0	1044

Figure 10: Locus Concordance by Fraction

Locus concordance by fraction

Genotype	discordance	nocall
ref-ref	0.00236	0.01628
het-ref-alt	0	0.05605
het-alt-alt	0	0
hom-alt-alt	0	0.03273
total	0.00099	0.03352

Figure 11: Locus Concordance by Count

Locus concordance by count				
Genotype	concordance	discordance	nocall	total
ref-ref	422	1	7	430
het-ref-alt	320	0	19	339
het-alt-alt	0	0	0	0
hom-alt-alt	266	0	9	275
total	1008	1	35	1044

Figure 12: Allele Concordance by Fraction

Allele concordance by fraction		
Genotype	discordance	nocall
ref-ref	0.00118	0.01395
het-ref-alt	0	0.0413
het-alt-alt	0	0
hom-alt-alt	0	0.01818
total	0.00049	0.02395

Figure 13: Allele Concordance by Fraction

Allele concordance by count				
Genotype	concordance	discordance	nocall	total
ref-ref	847	1	12	860
het-ref-alt	650	0	28	678
het-alt-alt	0	0	0	0
hom-alt-alt	540	0	10	550
total	2037	1	50	2088

Example

This example shows a comparison between the HapMap project to SNP calls made by Complete Genomics.

This example uses snpdiff to compare SNPs from a Yoruban female (NA19240, an individual genotyped in the HapMap project) to SNP calls from the same individual sequenced by Complete Genomics.

Input files are SNPs generated by the Illumina Infinium platform and the corresponding Complete Genomics variations file. The command produces three files:

- Output: lists a table of each compared SNP and their outcomes.
- Verbose: annotates the Output file with additional columns from the Complete Genomics ***variant*** file.
- Stats: summarizes statistics on the comparison.

```
cgatools snpdiff \
--genotypes /NA19240_HapMap_Infinium_37.tsv \
--variants /GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/var-GS19240-1100-37-ASM.tsv.bz2 \
--reference /ref/build37.crr \
--output-prefix NA19240_37_snpdiff_ \
--reports Output,Verbose,Stats
```

Data for NA19240 and others are available as part of the 69 public genomes dataset available at ftp2.completegenomics.com.

calldiff

Compares two variant files to determine where and how the two genomes differ. The following sections describe its behavior:

Synopsis

```
cgatools calldiff --help
       --variantsA <first_var_file>#<filter1>,<filter2>
       --variantsB <second_var_file>#<filter1>,<filter2>
       --reference <crr_file>
       --output-prefix <prefix>
       --reports <report_type>[,<report_type>...]
       --diploid
       --locus-stats column-count <count>
       --max-hypothesis-count <count>
       --no-reference-cover-validation
       --genome-rootA <directory>
       --genome-rootB <directory>
       --beta
       --calibration-root <arg>
```

Description

The calldiff tool compares two variant files to determine where and how the two genomes differ. To achieve this, it first gathers variants into superloci, which may concatenate several nearby variants. It compares the genomes for each superlocus then refines the comparison result to get call-level and locus-level detail.

calldiff is less sensitive than snpdiff to the canonical alignment. It achieves its specificity by being precise about its superlocus definition. For more information on the algorithm and how superloci are created, see "calldiff Algorithm" in the Appendix.

Analysis Pipeline Version Effects

- Analysis Pipeline version 2.4 introduced updated ***var*** and ***masterVarBeta*** file formats to provide new features such as ambiguous calling, minor allele frequencies from the 1000 Genomes Project, and changes to the variant quality flagging system. When operating on files generated before Analysis Pipeline 2.4, calldiff converts the output to the version 2.4 file format. It provides variant quality flags in the *varFilter* column (instead of *varQuality*) and adds *alleleFreq* and *alternativeCalls* columns whose fields are left empty.
- Analysis Pipeline version 2.0 introduced two independent scoring methods to quantify confidence in the reported call for each allele: Variable Allele Fraction (VAF) Scoring and Equal Allele Fraction (EAF) Scoring. The ***var*** and ***masterVarBeta*** files generated after 2.0 include the *varScoreVAF* and *varScoreEAF* fields. When operating on files generated before 2.0, calldiff fills *varScoreVAF* and *varScoreEAF* fields with the previously used metric *totalScore.* The *varFilter* field is left empty

Data Structure Requirement

The calldiff tool requires input files to be located in the directory hierarchy in which the data package was originally delivered by Complete Genomics.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--variantsA <first_var_file> #<filter1>,<filter2>`	Specifies the full or relative path to the "A" input variant file. Optionally you can include filters used to turn selected calls into no-calls. See the filtering syntax in the varfilter (beta) tool description.
`--variantsB <second_var_file> #<filter1>,<filter2>`	Specifies the full or relative path to the "B" input variant file. Optionally you can include filters used to turn selected calls into no-calls. See the filtering syntax in the varfilter (beta) tool description.
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file.
`--output-prefix <prefix>`	The path prefix for all output reports.
`--reports <report_type> [,<report_type>...]`	Comma-separated list of reports to generate. A report is one of `SuperlocusOutput`, `SuperlocusStats`, `LocusOutput`, `LocusStats`, `VariantOutput`, `SomaticOutput`, `DebugCallOutput`, `DebugSuperlocusOutput`, or `DebugSomaticOutput`. Table 5 describes the report types.
`--diploid`	Uses *varScoreEAF* instead of *varScoreVAF* in somatic score computations. Also, uses diploid variant model instead of variable allele mixture model.
`--locus-stats-column-count <count>`	The number of columns for locus compare classification in the locus statistics file. If this option is omitted, the column count defaults to 15.
`--max-hypothesis-count <count>`	The maximum number of possible phasings to consider for a superlocus. If this option is omitted, the count defaults to 32.
`--no-reference-cover-validation`	Turns off validation that all bases of a chromosome are covered by calls of the variant file.
`--genome-rootA <directory>`	The "A" genome directory.
`--genome-rootB <directory>`	The "B" genome directory, with similar expectations as A.
`--beta`	This flag enables the `SomaticOutput` report, which is beta functionality.
`--calibration-root <arg>`	The directory containing score calibration data. The directory should contain directories `version0.0.0` and `version2.0.0`. For example: `/home/complete/var-calibration-v1`

Table 5: calldiff Report Types

Report	Description
SuperlocusOutput	Superlocus classification.
SuperlocusStats	Superlocus classification statistics.
LocusOutput	Locus classification.
LocusStats	Locus statistics.
VariantOutput	Output includes columns from both variant files plus annotations from comparison results. If the SomaticOutput report is requested, the output includes the contents of the file specified by the `--variantsA` option with annotations from the same score ranks as produced in the SomaticOutput report.

Report	Description
SomaticOutput	List of simple variations that are present only in the file specified by the `--variantsA` option, annotated with the score that indicates the probability of the variation being truly somatic. To enable this report, you also need to specify the `--beta`, `--genome-rootA`, and `--genome-rootB` options. **Note:** Generating this report will cause calldiff to run 10 to 20 times as long as it would otherwise.
DebugCallOutput	Call classification.
DebugSuperlocusOutput	Debug superlocus information.
DebugSomaticOutput	Distribution estimates used for somatic rescoring. Only produced if SomaticOutput report is also specified.

Input Files

calldiff accepts two Complete Genomics variations files (***var-[ASM-ID].tsv.bz2***) or master variation files (***masterVarBeta-[ASM].tsv.bz2***) as input. These files can be in compressed form (.bz2). In addition, you can filter the input files to turn selected calls into no-calls, as described for varfilter (beta).

If the command line specifies the SomaticOutput report, the input needs to include calibration data for somatic score calculation. See "Obtaining a Reference Human Genome for Use with CGA Tools" for download instructions.

Output Files

calldiff produces various output files depending on the list of reports to generate that was specified on the command line. For each allele, calldiff generates a comparison classification value as described in Table 6. calldiff creates superloci for comparison and generates a variety of different reports that display the results of each comparison. For more information about superloci, see "calldiff Algorithm".

This section describes these reports:

- SuperlocusOutput
- SuperlocusStats
- LocusOutput
- LocusStats
- SomaticOutput

Table 6: Classification of Comparison Results

Classification	Description
ref-identical	The alleles of the two variant files are identical to each other and the reference.
alt-identical	The alleles of the two variant files are identical, but differ from the reference.
ref-consistent	Due to ambiguities (no-called bases), it is impossible to determine whether the alleles are identical. This classification indicates the alleles of the two variant files are consistent with each other and the reference.
alt-consistent	Due to ambiguities (no-called bases), it is impossible to determine whether the alleles are identical. This classification indicates the alleles of the two variant files are consistent with each other, but at least one allele differs from the reference.
onlyA	The alleles of the two variant files differ, and file A differs from the reference.
onlyB	The alleles of the two variant files differ, and file B differs from the reference.
Mismatch	The alleles of the two variant files differ with each other and the reference.
phase-mismatch	The two variant files would be consistent if the *hapLink* field had been empty, but the *hapLink* entry causes them to differ.
ploidy-mismatch	The superlocus did not have the same ploidy.

For non-haploid superloci, the comparison results for the alleles are joined by a semi-colon. For example, for a diploid hypothesis where variant file A calls reference and variant file B calls a het SNP, you will have the comparison result "ref-identical;onlyB".

For example, suppose we use calldiff to compare a tumor genome (file A) and a normal genome (file B) from the same individual. We can find purported somatic mutations by looking for "ref-identical;onlyA" or "onlyA;onlyA". Somatic mutations also include cases where tumor is homozygous variant and normal is homozygous reference. We can find purported loss of heterozygosity (LOH) by looking for "ref-identical;onlyB" (assuming the loss of the non-reference allele in the tumor) or "alt-identical;onlyA". We might expect fewer superloci classified as "alt-identical;onlyB", as the likely reason for this is assembly error: overcall in the normal genome.

SuperlocusOutput

Table 7: Columns in SuperlocusOutput Reports

	Column	Description
1	SuperlocusId	Superlocus identifier number
2	Chromosome	Chromosome name in text: `chr1`, `chr2`, …, `chr22`, `chrX`, `chrY`. The mitochondrial genome is represented as `chrM`. The pseudo-autosomal regions within the sex chromosomes X and Y are reported at their coordinates on chromosome X.
3	Begin	Reference coordinate specifying the start of the superlocus using the half-open, zero-based coordinate system.
4	End	Reference coordinate specifying the end of the superlocus using the half-open, zero-based coordinate system.
5	Classification	Results of the comparison. For non-haploid superloci, the results are joined by a semi-colon. See Table 6 for possible classification values.
6	Reference	The reference sequence for the locus of variation. This value is blank in the case of insertions.
7	AllelesA	The observed sequence at the locus of variation for both alleles from variant file A separated by a semi-colon.
8	AllelesB	The observed sequence at the locus of variation for both alleles from variant file B separated by a semi-colon.

SuperlocusStats

Figure 14: Example of a SuperlocusStats Report

SimpleClassification	Count	Pct
identical	32865	39.18%
consistent	12784	15.24%
onlyA	18291	21.80%
onlyB	19098	22.77%
mismatch	793	0.95%
phase-mismatch	54	0.06%
ploidy-mismatch	0	0.00%
Classification	Count	Pct
ref-identical;alt-identical	18242	21.75%
alt-identical;alt-identical	14623	17.43%
ref-identical;onlyB	12252	14.61%
ref-identical;onlyA	11903	14.19%
alt-identical;onlyB	5872	7.00%
ref-consistent;alt-consistent	5754	6.86%
alt-identical;onlyA	5506	6.56%
alt-identical;alt-consistent	2323	2.77%
alt-identical;ref-consistent	1954	2.33%
alt-consistent;alt-consistent	1623	1.93%
ref-identical;alt-consistent	1123	1.34%
ref-consistent;onlyB	756	0.90%
ref-consistent;onlyA	580	0.69%
alt-identical;mismatch	411	0.49%
alt-consistent;onlyA	274	0.33%
alt-consistent;onlyB	206	0.25%
ref-identical;mismatch	192	0.23%
ref-consistent;mismatch	93	0.11%
alt-consistent;mismatch	80	0.10%
ref-consistent;phase-mismatch	39	0.05%
onlyA;onlyA	28	0.03%
alt-consistent;phase-mismatch	15	0.02%
onlyB;onlyB	12	0.01%
mismatch;mismatch	6	0.01%
onlyA;mismatch	6	0.01%
ref-consistent;ref-consistent	6	0.01%
onlyB;mismatch	5	0.01%
ref-identical;ref-consistent	1	0.00%

Table 8: Columns in SuperlocusStats Reports

Column	Column Name	Description
Top 1	SimpleClassification	Possible classification values regardless of whether they are reference or alternate.
Top 2	Count	Number of observations for each SimpleClassification category.
Top 3	Pct	Percentage compared to the total Count of SimpleClassification values.
Bottom 1	Classification	All diploid permutations of possible Classification pairs except for cases where both alleles have the same Classification.
Bottom 2	Count	Number of observations for each Classification category.
Bottom 3	Pct	Percentage compared to the total Count of Classification values.

LocusOutput

Figure 15: Example of a LocusOutput Report

SuperlocusId	File	LocusClassification	LocusDiffClassification	locus	ploidy	allele
4796416	A	het-snp	ref-identical;alt-identical	20882706	2	1
4796416	A	het-snp	ref-identical;alt-identical	20882706	2	2
4796416	B	het-snp	ref-identical;alt-identical	21198798	2	1
4796416	B	het-snp	ref-identical;alt-identical	21198798	2	2
4796419	B	het-sub	ref-consistent;alt-consistent	21198804	2	1
4796419	B	het-sub	ref-consistent;alt-consistent	21198804	2	2
4796422	A	het-snp	ref-identical;onlyA	20882714	2	1
4796422	A	het-snp	ref-identical;onlyA	20882714	2	2

Figure 15: Example of a LocusOutput Report (continued)

SuperlocusId	chromosome	begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink	xRef	alleleFreq	alternativeCalls
4796416	chr21	9720985	9720986	ref	A	A	82	74	PASS				
4796416	chr21	9720985	9720986	snp	A	C	893	876	PASS		dbsnp.129:rs62218018		
4796416	chr21	9720985	9720986	ref	A	A	159	173	PASS	4543635			
4796416	chr21	9720985	9720986	snp	A	C	351	346	PASS	4543636	dbsnp.129:rs62218018		
4796419	chr21	9721062	9721064	ref	TG	TG	210	175	PASS				
4796419	chr21	9721062	9721064	sub	TG	CA	22	3	VQLOW				
4796422	chr21	9721116	9721117	ref	A	A	371	359	PASS				
4796422	chr21	9721116	9721117	snp	A	C	74	62	PASS		dbsnp.108:rs4068641		

Table 9: Columns in LocusOutput Reports

	Column	Description
1	SuperlocusId	Superlocus index.
2	File	Variant File A or Variant File B.
3	LocusClassification	Description of locus, whether it is a heterozygous, homozygous or no-call. Also includes information about the type of variation: snp, sub, indel.
4	LocusDiffClassification	Comparison results for each allele. Table 6 describes the types of classifications.
5	locus	Numerical index identifier of a particular locus from the Complete Genomics variant file (= 2 for autosomes, 2 for pseudoautosomal regions on the sex chromosomes, 1 for males on the non-pseudo-autosomal parts of the sex chromosomes, 1 for mitochondrion, 2 if *varType* is `no-ref` or `PARcalled-in-X`). The reported ploidy is fully determined by gender, chromosome and location, and is not inferred from the sequence data.
6	ploidy	Number of sets of the given variant in the genome.
7	haplotype	The allele that each variant belongs to.
8	chromosome	Chromosome name in text.
9	begin	Reference coordinate specifying the start of the variation (not the locus) using the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
10	end	Reference coordinate specifying the end of the variation (note the locus using the halfopen, zero-based coordinate system. See "Sequence Coordinate System" for more information.
11	varType	Type of variation, if any, for the range of bases. Currently must be one of snp, ins, del, sub, ref, no-call-rc, no-call-ri, no-call, No-ref, or PAR-called-in-X.
12	reference	The reference sequence for the locus of variation. Empty when *varType* is "ins". A value of "=" indicates that you must consult the reference for the sequence; this shorthand is only used in regions where no allele deviates from the reference sequence.
13	alleleSeq	The observed sequence at the locus of variation. Empty when *varType* is del. Question mark (?) indicates zero or more unknown bases within the sequence. "N" indicates exactly one unknown base within the sequence. Equal sign (=) is used as shorthand to indicate identity to the reference sequence for non-variant sequence, such as when *varType* is ref.
14	varScoreVAF	Variable allele fraction model confidence score from the Complete Genomics variation file (*varScoreVAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is populated with the *totalScore*. This field is empty for reference calls or no-calls.
15	varScoreEAF	Equal allele fraction model confidence score from the Complete Genomics variation file (*varScoreEAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is equivalent to the *totalScore*. This field is empty for reference calls or no-calls.
16	varFilter	List of indicators of low-quality or incomplete resolution of the variant call. If "PASS", then the allele passes all relevant quality tests. Otherwise, the list includes one or more semicolon-separated values from the following possible filters: ▪ `VQLOW` — Indicates the call is homozygous and *allele1VarScoreVAF* is less than 20 dB, or the call is not homozygous and *allele1VarScoreVAF* is less than 40 dB. ▪ `SQLOW` — Indicates somatic variant has *somaticScore* < -10. ▪ `AMBIGUOUS` — For homozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 10 dB of the call; for heterozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 20 dB of the call.

	Column	Description
17	hapLink	Identifier that links an allele at one locus to alleles at other loci. Currently only populated for very proximate variations that were either assembled together or were determined to be in phase using a correlation-based analysis between two variation intervals one mate pair away. Calls that share a hapLink identifier are expected to be on the same haplotype. Calls with haplinks appearing only once in the file and calls with no haplinks can be interpreted similarly: there is no phasing information with any other loci.
18	xRef	Field containing external variation identifiers, populated for variations corroborated directly by dbSNP and COSMIC. Format for dbSNP: `dbsnp.<build>:<rsID>`, with multiple entries separated by the semicolon (;). `<build>` indicates in which build of dbSNP this entry first appeared. For example, "dbsnp.129:rs12345". Format for COSMIC: `COSMIC.<type>:identifier`, with multiple entries separated by the semicolon (;). `<type>` indicates COSMIC classification of somatic variants. For example for a non-coding variant, *xRef* would contain "COSMIC:ncv_id:139111".
19	alleleFreq	Allele frequency value(s) for the entire call or for parts of the call that are corroborated directly by external sources. The source is 1000 Genomes Project minor allele frequency information in dbSNP. Format is `<source>:<frequency>`, with multiple pairs separated by a semicolon. Precision of `frequency` is three decimal places. Format for dbSNP becomes `dbsnp:<frequency>`. Multiple entries for the same type of source mirror the multiple entries for this source appearing in *xRef*. If an allele frequency value is not available for a dbSNP record, the corresponding position in the *alleleFreq* column is left empty.
20	alternativeCalls	Contains alternate calls for alleles designated "`AMBIGUOUS`". Formatted as a semicolon-separated list of `<sequence>:<score>` pairs, where `<sequence>` is a hypothesized nucleotide sequence, and `<score>` is the score of that hypothesized sequence, relative to the called sequence. For example, if *alternativeCalls* is "`AG:-1;G:-8`", then sequence AG scored 1 dB less than the called sequence and G scored 8 dB less than the called sequence.

When the call matches...	The *alleleFreq* field shows...
1 rsID with known frequency.	`dbsnp:0.234`
1 rsID with unknown frequency.	(empty string)
2 rsIDs (independently or as a combination) with both known frequencies.	`dbsnp:0.234;dbsnp:0.123`
2 rsIDs with both unknown frequencies.	`;` (a single semicolon)

3 rsIDs with 1 known and 2 unknown frequencies.	Depending on which of the frequencies are unknown, one of the following:
	`dbsnp:0.234;;`
	`;dbsnp:0.234;`
	`;;dbsnp:0.234`

LocusStats

A single file that provides summary statistics for each combination of locus class and the type of call made, as shown in the following figures.

Figure 16: Locus Statistics: File A by Percent of LocusClassification

Locus stats for file A by pct of LocusClassification

File	LocusClassification	ref-identical;alt-identical	alt-identical;alt-identical	ref-identical;onlyB	ref-identical;onlyA	alt-identical;onlyB	alt-identical;onlyA	ref-consistent;alt-consistent	alt-identical;ref-consistent	alt-identical;alt-consistent
A	het-del	42.13%	0.00%	0.00%	31.14%	10.49%	0.00%	5.97%	3.18%	0.00%
A	het-ins	42.87%	0.00%	0.00%	26.44%	10.80%	0.00%	5.56%	4.52%	0.00%
A	het-other	0.00%	28.32%	0.00%	0.00%	2.15%	24.61%	0.00%	0.00%	14.26%
A	het-snp	48.18%	0.00%	0.00%	29.95%	14.25%	0.00%	4.41%	1.74%	0.00%
A	het-sub	38.48%	0.00%	0.00%	27.62%	13.74%	0.00%	12.04%	1.70%	0.00%
A	hom-del	0.00%	63.38%	0.00%	0.00%	0.00%	18.67%	0.00%	0.00%	10.22%
A	hom-ins	0.00%	71.95%	0.00%	0.00%	0.00%	12.21%	0.00%	0.00%	10.25%
A	hom-other	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
A	hom-snp	0.00%	66.89%	0.00%	0.00%	0.00%	25.11%	0.00%	0.00%	4.51%
A	hom-sub	0.00%	59.43%	0.00%	0.00%	0.00%	23.46%	0.00%	0.00%	9.21%
A	no-call-del	0.00%	0.00%	0.00%	0.00%	1.67%	0.00%	43.53%	43.38%	0.00%
A	no-call-ins	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	41.68%	48.68%	0.00%
A	no-call-no-call	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	92.05%	0.00%	0.00%
A	no-call-other	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	38.68%	46.23%	5.66%
A	no-call-snp	0.00%	0.00%	0.00%	0.00%	0.70%	0.00%	27.41%	60.68%	0.00%
A	no-call-sub	0.00%	0.00%	0.00%	0.00%	2.99%	0.00%	34.33%	44.03%	0.00%
A	total	28.89%	22.56%	0.00%	18.13%	8.54%	8.21%	4.95%	3.52%	1.85%

Figure 16: Locus Statistics: File A by Percent of LocusClassification (continued)

File	LocusClassification	ref-identical;alt-consistent	alt-consistent;alt-consistent	ref-consistent;onlyB	ref-consistent;onlyA	alt-consistent;onlyA	alt-identical;mismatch	other	total
A	het-del	5.52%	0.00%	0.00%	0.39%	0.00%	0.00%	1.17%	100.00%
A	het-ins	9.16%	0.00%	0.00%	0.13%	0.00%	0.00%	0.52%	100.00%
A	het-oth	0.00%	13.28%	0.00%	0.00%	4.88%	10.35%	2.15%	100.00%
A	het-snp	1.36%	0.00%	0.00%	0.01%	0.00%	0.00%	0.10%	100.00%
A	het-sub	4.58%	0.00%	0.00%	0.65%	0.00%	0.00%	1.18%	100.00%
A	hom-del	0.00%	4.53%	0.00%	0.00%	1.96%	0.36%	0.89%	100.00%
A	hom-ins	0.00%	3.92%	0.00%	0.00%	1.31%	0.15%	0.22%	100.00%
A	hom-oth	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
A	hom-snp	0.00%	2.71%	0.00%	0.00%	0.61%	0.10%	0.08%	100.00%
A	hom-sub	0.00%	3.29%	0.00%	0.00%	1.75%	1.32%	1.54%	100.00%
A	no-call	0.00%	0.00%	0.00%	8.37%	0.00%	0.00%	3.04%	100.00%
A	no-call	0.00%	0.00%	0.00%	8.71%	0.00%	0.00%	0.93%	100.00%
A	no-call	0.00%	0.44%	0.22%	5.52%	0.00%	0.00%	1.77%	100.00%
A	no-call	0.00%	2.83%	0.00%	3.77%	0.00%	0.94%	1.89%	100.00%
A	no-call	0.00%	0.00%	0.13%	11.03%	0.00%	0.00%	0.06%	100.00%
A	no-call	0.00%	0.00%	0.00%	12.69%	0.00%	0.00%	5.97%	100.00%
A	total	1.14%	1.07%	0.00%	0.49%	0.28%	0.12%	0.24%	100.00%

Figure 17: Locus Statistics: File B by Percent of LocusClassification

Locus stats for file B by pct of LocusClassification

File	LocusClassification	ref-identical;alt-identical	alt-identical;alt-identical	ref-identical;onlyB	ref-identical;onlyA	alt-identical;onlyB	alt-identical;onlyA	ref-consistent;alt-consistent	alt-identical;ref-consistent	alt-identical;alt-consistent
B	het-del	42.51%	0.00%	33.94%	0.00%	0.00%	10.80%	4.40%	2.17%	0.00%
B	het-ins	44.64%	0.00%	28.24%	0.00%	0.00%	10.59%	4.22%	4.08%	0.00%
B	het-other	0.00%	27.74%	0.00%	0.00%	30.94%	3.58%	0.19%	0.19%	15.28%
B	het-snp	49.63%	0.00%	31.93%	0.00%	0.00%	13.77%	2.69%	0.96%	0.00%
B	het-sub	40.73%	0.00%	29.71%	0.00%	0.00%	14.50%	6.69%	1.39%	0.00%
B	hom-del	0.00%	67.39%	0.00%	0.00%	18.66%	0.00%	0.00%	0.00%	8.48%
B	hom-ins	0.00%	74.77%	0.00%	0.00%	13.55%	0.00%	0.00%	0.00%	8.28%
B	hom-other	0.00%	0.00%	0.00%	0.00%	50.00%	0.00%	0.00%	0.00%	50.00%
B	hom-snp	0.00%	68.16%	0.00%	0.00%	27.32%	0.00%	0.00%	0.00%	2.50%
B	hom-sub	0.00%	67.08%	0.00%	0.00%	23.27%	0.00%	0.00%	0.00%	5.20%
B	no-call-del	0.00%	0.00%	0.00%	0.00%	0.00%	1.08%	36.79%	51.00%	0.00%
B	no-call-ins	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	35.09%	54.23%	0.00%
B	no-call-no-call	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	90.33%	0.00%	0.00%
B	no-call-no-call-ri	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%
B	no-call-other	0.00%	0.00%	0.00%	0.00%	0.67%	0.00%	26.00%	62.00%	1.33%
B	no-call-snp	0.00%	0.00%	0.00%	0.00%	0.00%	0.81%	15.82%	69.50%	0.00%
B	no-call-sub	0.00%	0.00%	0.00%	0.00%	0.00%	2.24%	26.87%	54.48%	0.00%
B	total	29.30%	22.94%	19.04%	0.00%	8.88%	8.17%	3.71%	4.12%	1.17%

Figure 17: Locus Statistics: File B by Percent of LocusClassification (continued)

File	LocusClassification	ref-identical;alt-consistent	alt-consistent;alt-consistent	ref-consistent;onlyB	ref-consistent;onlyA	alt-consistent;onlyA	alt-identical;mismatch	other	total
B	het-de	5.09%	0.00%	0.11%	0.00%	0.06%	0.00%	0.91%	100.00%
B	het-in	7.61%	0.00%	0.28%	0.00%	0.00%	0.00%	0.35%	100.00%
B	het-ot	0.00%	6.79%	0.19%	0.00%	0.00%	10.00%	5.09%	100.00%
B	het-sn	0.91%	0.00%	0.01%	0.00%	0.01%	0.00%	0.08%	100.00%
B	het-su	4.74%	0.00%	0.56%	0.00%	0.00%	0.00%	1.67%	100.00%
B	hom-de	0.00%	2.83%	0.00%	0.00%	0.00%	0.19%	2.45%	100.00%
B	hom-in	0.00%	2.41%	0.00%	0.00%	0.00%	0.15%	0.83%	100.00%
B	hom-ot	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
B	hom-sn	0.00%	1.58%	0.00%	0.00%	0.00%	0.08%	0.35%	100.00%
B	hom-su	0.00%	2.23%	0.00%	0.00%	0.00%	1.24%	0.99%	100.00%
B	no-cal	0.00%	0.00%	6.96%	0.00%	0.15%	0.15%	3.86%	100.00%
B	no-cal	0.00%	0.00%	9.57%	0.16%	0.00%	0.00%	0.96%	100.00%
B	no-cal	0.00%	0.56%	7.43%	0.37%	0.37%	0.00%	0.93%	100.00%
B	no-cal	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
B	no-cal	0.00%	3.33%	5.33%	0.00%	0.00%	0.67%	0.67%	100.00%
B	no-cal	0.00%	0.00%	#####	0.14%	0.05%	0.00%	0.50%	100.00%
B	no-cal	0.00%	0.00%	#####	0.00%	0.00%	0.00%	2.99%	100.00%
B	total	0.83%	0.63%	0.69%	0.01%	0.01%	0.12%	0.37%	100.00%

Figure 18: Locus Statistics: File A by Count

Locus stats for file A by count

File	LocusClassification	ref-identical;alt-identical	alt-identical;alt-identical	ref-identical;onlyB	ref-identical;onlyA	alt-identical;onlyB	alt-identical;onlyA	ref-consistent;alt-consistent	alt-identical;ref-consistent	alt-identical;alt-consistent	ref-identical;alt-consistent	alt-consistent;alt-consistent	ref-consistent;onlyB	ref-consistent;onlyA	alt-consistent;onlyA	alt-identical;mismatch	other	total
A	het-del	755	0	0	558	188	0	107	57	0	99	0	0	7	0	0	21	1792
A	het-ins	655	0	0	404	165	0	85	69	0	140	0	0	2	0	0	8	1528
A	het-other	0	145	0	0	11	126	0	0	73	0	68	0	0	25	53	11	512
A	het-snp	18549	0	0	11532	5488	0	1698	669	0	522	0	0	2	0	0	40	38500
A	het-sub	294	0	0	211	105	0	92	13	0	35	0	0	5	0	0	9	764
A	hom-del	0	713	0	0	0	210	0	0	115	0	51	0	0	22	4	10	1125
A	hom-ins	0	990	0	0	0	168	0	0	141	0	54	0	0	18	2	3	1376
A	hom-other	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
A	hom-snp	0	13696	0	0	0	5141	0	0	923	0	555	0	0	125	20	16	20476
A	hom-sub	0	271	0	0	0	107	0	0	42	0	15	0	0	8	6	7	456
A	no-call-del	0	0	0	0	11	0	286	285	0	0	0	0	55	0	0	20	657
A	no-call-ins	0	0	0	0	0	0	268	313	0	0	0	0	56	0	0	6	643
A	no-call-no-call	0	0	0	0	0	0	417	0	0	0	2	1	25	0	0	8	453
A	no-call-other	0	0	0	0	0	0	41	49	6	0	3	0	4	0	1	2	106
A	no-call-snp	0	0	0	0	11	0	430	952	0	0	0	2	173	0	0	1	1569
A	no-call-sub	0	0	0	0	4	0	46	59	0	0	0	0	17	0	0	8	134
A	total	20253	15816	0	12705	5983	5752	3470	2466	1300	796	748	3	346	198	86	170	70092

Figure 19: Locus Statistics: File B by Count

Locus stats for file B by count

File	LocusClassification	ref-identical;alt-identical	alt-identical;alt-identical	ref-identical;onlyB	ref-identical;onlyA	alt-identical;onlyB	alt-identical;onlyA	ref-consistent;alt-consistent	alt-identical;ref-consistent	alt-identical;alt-consistent	ref-identical;alt-consistent	alt-consistent;alt-consistent	ref-consistent;onlyB	ref-consistent;onlyA	alt-consistent;onlyA	alt-identical;mismatch	other	total
B	het-del	744	0	594	0	0	189	77	38	0	89	0	2	0	1	0	16	1750
B	het-ins	645	0	408	0	0	153	61	59	0	110	0	4	0	0	0	5	1445
B	het-other	0	147	0	0	164	19	1	1	81	0	36	1	0	0	53	27	530
B	het-snp	18521	0	11915	0	0	5139	1004	360	0	341	0	4	0	3	0	31	37318
B	het-sub	292	0	213	0	0	104	48	10	0	34	0	4	0	0	0	12	717
B	hom-del	0	715	0	0	198	0	0	0	90	0	30	0	0	0	2	26	1061
B	hom-ins	0	993	0	0	180	0	0	0	110	0	32	0	0	0	2	11	1328
B	hom-other	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2
B	hom-snp	0	13689	0	0	5487	0	0	0	502	0	318	0	0	0	17	70	20083
B	hom-sub	0	271	0	0	94	0	0	0	21	0	9	0	0	0	5	4	404
B	no-call-del	0	0	0	0	0	7	238	330	0	0	0	45	0	1	1	25	647
B	no-call-ins	0	0	0	0	0	0	220	340	0	0	0	60	1	0	0	6	627
B	no-call-no-call	0	0	0	0	0	0	486	0	0	0	3	40	2	2	0	5	538
B	no-call-no-call-ri	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
B	no-call-other	0	0	0	0	1	0	39	93	2	0	5	8	0	0	1	1	150
B	no-call-snp	0	0	0	0	0	18	350	1538	0	0	0	292	3	1	0	11	2213
B	no-call-sub	0	0	0	0	0	3	36	73	0	0	0	18	0	0	0	4	134
B	total	20202	15815	13130	0	6125	5632	2561	2842	807	574	433	478	6	8	81	254	68948

SomaticOutput

Somatic variation discovery is an important use case for calldiff. calldiff will identify variations that exist only in the input file A and not in file B and assign a somatic score to each of those variants to help identify the true somatic mutations. calldiff uses the scores provided in the Complete Genomics variation file (*varScoreVAF*, or *totalScores* for data prior to Assembly 2.0) and the calibrated scores specified by the user to determine which somatic mutations are called with higher confidence, and provides this information as a single somatic score. It does so for all loci where the genome A has a simple variation (a single SNP, DEL, INS, or SUB) and genome B is called as reference.

To aid in filtering for high quality variants, the SomaticOutput report includes three values:

- *SomaticScore*: integer that provides a rank order on somatic call confidence across all somatic variation types specified in *SomaticCategory*.
- *SomaticRank*: number between 0 and 1 indicating the estimated fraction of events within the *SomaticCategory* having *SomaticScore* less than that of the variant. It can be used to estimate the sensitivity trade-off when applying a *SomaticScore* threshold.

- *varFilter*: includes SQLOW value, which indicates low confidence in the somatic call.

For more information on how these fields are computed, see "calldiff for Scoring Somatic Variations (beta)" in the Appendix.

To run calldiff with the SomaticOutput option, the `--genome-rootA` and `--genome-rootB` options must also be specified, pointing to calibration data. You can download the calibration data from Complete Genomics' FTP site. See "Obtaining Ancillary Files for Use with CGA Tools" for instructions.

Figure 20: Example of SomaticOutput Report

SuperlocusId	LocusClassification	locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink
12	het-snp	992	2	2	chr1	49403	49404	snp	C	T	33	9	VQLOW;SQLOW	
51	het-del	1234	2	2	chr1	64657	64658	del	A		41	29	SQLOW	
258	het-snp	12588	2	2	chr1	731685	731686	snp	G	A	29	-14	VQLOW;SQLOW	
304	het-ins	13138	2	2	chr1	770312	770312	ins		ACCT	570	583	PASS	4906
344	het-snp	13452	2	2	chr1	794703	794704	snp	C	T	412	412	PASS	

Figure 20: Example of SomaticOutput Report (continued)

SuperlocusId	xRef	alleleFreq	alternativeCalls	VarCvgA	VarScoreA	RefCvgB	RefScoreB	SomaticCategory	VarScoreACalib	RefScoreBCalib	SomaticRank	SomaticScore
12	dbsnp.100:rs2531246;dbsnp.130:rs74182487	;		96	33	25	-21	snp	2	41	0.003	-28
51	dbsnp.130:rs71333268			36	41	40	38	del	8	68	0.053	-21
258	dbsnp.101:rs2995839			77	29	81	81	snp	1	57	0.002	-29
304				97	570	45	176	ins	36	77	0.941	4
344				58	412	73	129	snp	47	71	0.314	10

Table 10: Column Descriptions from SomaticOutput Report

	Column	Description
1	SuperlocusId	Superlocus index.
2	LocusClassification	Description of locus, whether it is a heterozygous, homozygous or no-call. Also includes information about the type of variation: snp, sub, indel.
3	locus	Numerical index identifier of a particular locus from the Complete Genomics variant file.
4	ploidy	The ploidy of the reference genome at the locus (= 2 for autosomes, 2 for pseudo-autosomal regions on the sex chromosomes, 1 for males on the non-pseudo-autosomal parts of the sex chromosomes, 1 for mitochondrion, 2 if *varType* is `no-ref` or `PAR-called-in-X`). The reported ploidy is fully determined by gender, chromosome and location, and is not inferred from the sequence data.
5	allele	Identifier for each allele at the variation locus.
6	chromosome	Chromosome name.

	Column	Description
7	begin	Reference coordinate specifying the start of the variation using the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
8	end	Reference coordinate specifying the end of the variation using the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
9	varType	Type of variation for the range of bases. Currently must be one of `snp`, `ins`, `del`, or `sub`.
10	reference	The reference sequence for the locus of variation. Empty when *varType* is `ins`.
11	alleleSeq	The observed sequence at the locus of variation. Empty when *varType* is `del`.
12	varScoreVAF	Variable allele fraction model confidence score from the Complete Genomics variation file (*varScoreVAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is populated with the *totalScore*. This field is empty for reference calls or no-calls.
13	varScoreEAF	Equal allele fraction model confidence score from the Complete Genomics variation file (*varScoreEAF* column). For genomes assembled before Analysis Pipeline software version 2.0, this field is equivalent to the *totalScore*. This field is empty for reference calls or no-calls.
14	varFilter	List of indicators of low-quality or incomplete resolution of the variant call. If "PASS", then the allele passes all relevant quality tests. Otherwise, the list includes one or more semicolon-separated values from the following possible filters: ▪ `VQLOW` — Indicates the call is homozygous and *allele1VarScoreVAF* is less than 20 dB, or the call is not homozygous and *allele1VarScoreVAF* is less than 40 dB. ▪ `SQLOW` — Indicates somatic variant has *somaticScore* < -10. ▪ `AMBIGUOUS` — For homozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 10 dB of the call; for heterozygous non-reference alleles, indicates there was another non-reference hypothesized sequence that scored within 20 dB of the call.
15	hapLink	Identifier that links an allele at one locus to alleles at other loci. Currently only populated for very proximate variations that were either assembled together or were determined to be in phase using a correlation-based analysis between two variation intervals one mate pair away. Calls that share a hapLink identifier are expected to be on the same haplotype. Calls with haplinks appearing only once in the file and calls with no haplinks can be interpreted similarly: there is no phasing information with any other loci.
16	xRef	Field containing external variation identifiers, populated for variations corroborated directly by dbSNP and COSMIC. Format for dbSNP: `dbsnp.<build>:<rsID>`, with multiple entries separated by the semicolon (;). `<build>` indicates in which build of dbSNP this entry first appeared. For example, `dbsnp.129:rs12345`. Format for COSMIC: `COSMIC.<type>:identifier`, with multiple entries separated by the semicolon (;). `<type>` indicates COSMIC classification of somatic variants. For example for a non-coding variant, *xRef* would contain "COSMIC:ncv_id:139111".

<table>
<tr><th></th><th>Column</th><th>Description</th></tr>
<tr><td>17</td><td>alleleFreq</td><td>Allele frequency value(s) for the entire call or for parts of the call that are corroborated directly by external sources. The source is 1000 Genomes Project minor allele frequency information in dbSNP.

Format is <source>:<frequency>, with multiple pairs separated by a semicolon. Precision of frequency is three decimal places.

Format for dbSNP becomes dbsnp:<frequency>. Multiple entries for the same type of source mirror the multiple entries for this source appearing in xRef.

If an allele frequency value is not available for a dbSNP record, the corresponding position in the alleleFreq column is left empty.

<table>
<tr><th>When the call matches...</th><th>The alleleFreq field shows...</th></tr>
<tr><td>1 rsID with known frequency.</td><td>dbsnp:0.234</td></tr>
<tr><td>1 rsID with unknown frequency.</td><td>(empty string)</td></tr>
<tr><td>2 rsIDs (independently or as a combination) with both known frequencies.</td><td>dbsnp:0.234;dbsnp:0.123</td></tr>
<tr><td>2 rsIDs with both unknown frequencies.</td><td>; (a single semicolon)</td></tr>
<tr><td>3 rsIDs with 1 known and 2 unknown frequencies.</td><td>Depending on which of the frequencies are unknown, one of the following:

dbsnp:0.234;;
;dbsnp:0.234;
;;dbsnp:0.234</td></tr>
</table>
</td></tr>
<tr><td>18</td><td>alternativeCalls</td><td>Contains alternate calls for alleles designated “AMBIGUOUS”. Formatted as a semicolon-separated list of <sequence>:<score> pairs, where <sequence> is a hypothesized nucleotide sequence, and <score> is the score of that hypothesized sequence, relative to the called sequence.

For example, if alternativeCalls is “AG:-1;G:-8”, then sequence AG scored 1 dB less than the called sequence and G scored 8 dB less than the called sequence.</td></tr>
<tr><td>19</td><td>VarCvgA</td><td>The totalReadCount from the A genome masterVarBeta file (or if the A assembly is a var file, the equivalent is computed by calldiff).</td></tr>
<tr><td>20</td><td>VarScoreA</td><td>Equal to varScoreVAF of the A genome by default, or varScoreEAF if the --diploid option is used.</td></tr>
<tr><td>21</td><td>RefCvgB</td><td>The uniqueSequenceCoverage field from the reference scores file for genome B at this locus.</td></tr>
<tr><td>22</td><td>RefScoreB</td><td>The refScore field from the reference scores file for genome B at this locus.</td></tr>
<tr><td>23</td><td>SomaticCategory</td><td>The category of this mutation. Possible categories are: snp, ins, del, and sub. The somaticRank is described with respect to all mutations in the somaticCategory.</td></tr>
<tr><td>24</td><td>VarScoreACalib</td><td>The calibrated VarScoreA under the allele fraction model defined by use of the --diploid option and corrected for the count of events in this genome.</td></tr>
<tr><td>25</td><td>RefScoreBCalib</td><td>The calibrated RefScoreB under the allele fraction model defined by the use of the --diploid option and corrected for the count of events in this genome.</td></tr>
<tr><td>26</td><td>SomaticRank</td><td>The estimated rank of this somatic mutation, amongst all true somatic mutations within a given somaticCategory. Value is a number between 0 and 1; a value of 0.012 means, for example, that 1.2% of the true somatic mutations in this somaticCategory have a somaticScore less than the somaticScore for this mutation.</td></tr>
<tr><td>27</td><td>SomaticScore</td><td>An integer that provides a total order on quality for all somatic mutations. It is equal to
$$-10 * log10(P(false)/P(true))$$
under the assumption that this genome has a rate of somatic mutation equal to 1/Mb for somaticCategory snp, 1/10Mb for somaticCategory ins, 1/10Mb for somaticCategory del, and 1/20Mb for somaticCategory sub.</td></tr>
</table>

Examples

Gene LIPI and Variation

You are interested in studying the gene LIPI and its variation in two Complete Genomics-sequenced individuals. You know that the LIPI gene is located at chr21:15481137-15579254 (UCSC coordinates).

This example shows a comparison of variants from two Complete Genomics-sequenced genomes (GSXXXXX and the second is GSYYYYY) using calldiff.

```
cgatools calldiff \
--variantsA /GSXXXXX-DNA_A01_1120_37-ASM/GSXXXXX-DNA_A01/ASM/var-GSXXXXX-1100-37-ASM.tsv.bz2 \
--variantsB /GSYYYYY-DNA_A01_1120_37-ASM/GSYYYYY-DNA_A01/ASM/var-GSYYYYY-1100-37-ASM.tsv.bz2 \
--reference ref/build37.crr \
--output-prefix GSXXXXX_vs_GSYYYYY \
--reports SuperlocusOutput,SuperlocusStats,LocusOutput,LocusStats
```

In the SuperlocusOutput file, you would look for the all Superloci located between the boundaries of the LIPI gene (chr21:15481137-15579254). These are the variants in the two genomes that were explicitly defined into superloci for comparison purposes. You can then look over the output to discover loci that are different in the two genomes. These will typically be defined classified as "onlyA or onlyB" indicating that the variant was found in only one of the two genomes. The superlocus output will also provide information on whether alleles were consistent with the reference sequence or not.

High-confidence Somatic Variants

You are interesting in finding a high-confidence set of somatic variants in a tumor sample that are not present in a normal sample.

This example shows a comparison of a tumor genome to a matched normal sample using the SomaticOutput option in calldiff.

```
cgatools calldiff \
--beta \
--variantsA GSXXXXX_tumor/ASM/GSXXXXX_tumor.tsv.bz2 \
--variantsB GSYYYYY_normal/ASM/GSXXXXX_normal.tsv.bz2 \
--reference /ref/build37.crr \
--calibration-root /ref/calibration-data \
--output-prefix tumor_vs_normal_ \
--reports SuperlocusOutput,SuperlocusStats,LocusOutput,LocusStats,SomaticOutput \
--genome-rootA GSXXXXX_tumor/GSXXXXX-DNA_C01/ASM/GSXXXXX-DNA_A01 \
--genome-rootB GSYYYYY_normal/GSYYYYY-DNA_D01/ASM/GSYYYYY-DNA_C01
```

Note that the report for the SomaticOutput option will take approximately four hours to generate, depending on computing power and RAM and assuming the input variant files are ***masterVar*** files. It may take substantially longer to generate SomaticOutput for ***var*** files because the *totalReadCount* must be re-computed from EVIDENCE files.

In the resulting SomaticOutput report file, you can then rank the somatic score column in descending order. Higher scores indicate a greater likelihood that there is indeed a somatic variation between the two samples at this position (as opposed to a false positive).

listvariants (beta)

Lists the non-redundant set of small variations found in an arbitrary number of genomes. The following sections describe its behavior:

Synopsis

```
cgatools listvariants --help
    --beta
    --reference <crr_file>
    –-output <output_file>
    --variants <variant_fileA>#<filter1>,<filter2>
    --variants <variant_fileB>#<filter1>,<filter2>
    --variant-listing <previous_output>
    --list-long-variants
```

Description

The listvariants command lists the non-redundant set of small variations found in an arbitrary number of genomes. It takes an arbitrary number of ***var*** or ***masterVar*** files as input, and produces a tab-delimited format suitable for processing by the testvariants command. listvariants may be used in conjunction with testvariants to perform multi-genome comparison of small variants.

Analysis Pipeline Version Effects

listvariants in CGA Tools version 1.5 was altered to support new canonicalization rules used for variant calling in Analysis Pipeline version 2.0. Specifically, Analysis Pipeline version 2.0 uses a left-most canonicalization of indels, whereas previous versions use a right-most canonicalization rule. To avoid re-canonicalization of a large number of variants when running listvariants, we recommend that comparison of ***var*** or ***masterVarBeta*** files from Pipeline versions earlier than 2.0 be performed using listvariants in CGA Tools 1.4.

For more information about how listvariants combines small variants across multiple genomes, see "listvariants Algorithm" in the Appendix.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables listvariants (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.
`--output <output_file>`	The output file. If this option is omitted, results are sent to `STDOUT`.

Option	Description
`--variants <variant_file> #<filter1>,<filter2>`	This option can be used once for each input variant file. Files can be ***var*** or ***masterVar*** files. Optionally you can include filters used to turn selected calls into no-calls. See the filtering syntax in the varfilter (beta) tool description. Alternatively, you can read variant file names from a text file. The files must be specified with full path names, separated by spaces, and listed on a single line. For example, to read a list of files from dir_output.txt as the argument for the `--variants` option: `--variants \`cat dir_output.txt\`` Note that the argument is enclosed with backticks rather than single quotes.
`--variant-listing <previous_output>`	The full or relative path to a file containing the output of another listvariants run, to be merged in to produce the output of this run.
`--list-long-variants`	List longer variants (10's of bases) in addition to listing short variants by concatenating nearby calls.

Input Files

listvariants takes a reference CRR file of the appropriate build and a set of two or more Complete Genomics variations files (***var-[ASM-ID].tsv.bz2***) or ***masterVar*** file (***masterVarBeta-[ASM-ID].tsv.bz2***) as input. These files may be in uncompressed or compressed (.bz2) form. In addition, you can filter the input files to turn selected calls into no-calls, as described for varfilter (beta).

Output Files

The listvariants output file retains the external reference (*xRef*) annotations present in the input variant file. Figure 21 shows an example of the listvariants output; the columns are described in Table 11.

Figure 21: Example listvariants Output

variantId	chromosome	begin	end	varType	reference	alleleSeq	xRef
1034	chr1	972803	972804	snp	T	C	dbsnp:rs3128102
1035	chr1	972856	972857	snp	T	C	dbsnp:rs10267
1036	chr1	975024	975025	snp	G	T	
1037	chr1	975128	975129	snp	C	T	dbsnp:rs2275813
1038	chr1	975311	975313	sub	GG	A	dbsnp:rs56255212
1039	chr1	975322	975323	snp	T	C	dbsnp:rs2275811
1040	chr1	975371	975372	snp	G	A	
1041	chr1	975900	975901	snp	G	A	

Table 11: Column Descriptions for listvariants Output

	Column Name	Description
1	variantId	Sequential ID assigned to each variant.
2	chromosome	The chromosome of the variant.
3	begin	Zero-based reference offset of the beginning of the variant.
4	end	Zero-based reference offset of the end of the variant.
5	varType	The *varType* as extracted from the variant file.
6	reference	The reference sequence.
7	alleleSeq	The variant allele sequence as extracted from the variant file.
8	xRef	The *xRef* as extracted from the variant file.

Examples

List Multiple Genomes on the Command Line

This command creates a file called `listvariants_YRI_trio.tsv` that contains the variants present in the NA19238, NA19239, and NA19240 genomes.

```
cgatools listvariants \
--beta \
--reference /complete/build37.crr \
--output /complete/listvariants_YRI_trio.tsv \
--variants \
/complete/GS19238-1100-37-ASM/GS00028-DNA_A01/ASM/var-GS19238-1100-37-ASM.tsv.bz2 \
/complete/GS19239-1100-37-ASM/GS00028-DNA_B01/ASM/var-GS19239-1100-37-ASM.tsv.bz2 \
/complete/GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/var-GS19240-1100-37-ASM.tsv.bz2
```

List Multiple Genomes in a Text File

This command creates a file called `listvariants_YRI_trio.tsv` that contains the variants present in the NA19238, NA19239, and NA19240 genomes. In this example, the paths to the variation files are contained in a text file called `listvariant_input.txt` and the cat command is used to provide the variations to listvariants. listvariants_input.txt should contain the name and full paths to all input genome variation files **on one line**, separated by spaces.

The input file (in this case, `listvariant_input.txt`) should list the full paths and filenames for all variant files, separated by spaces, on a single line. For example:

```
/complete/var-GS19238-1100-37-ASM.tsv.bz2 /complete/var-GS19239-1100-37-ASM.tsv.bz2
/complete/var-GS19240-1100-37-ASM.tsv.bz2
```

```
cgatools listvariants \
--beta \
--reference /complete/build37.crr \
--output /complete/listvariants_YRI_trio.tsv \
--variants `cat /complete/listvariants_input.txt`
```

Important: Make sure to use backticks (`) instead of single quotes.

testvariants (beta)

Determine which variants are found in which genomes given the results of listvariants. The following sections describe its behavior:

Synopsis

```
cgatools testvariants --help
       --beta
       --reference <crr_file>
       --input <listvariants_output>
       –-output <output_file>
       --variants <variant_fileA>#<filter1>,<filter2>
--variants <variant_fileB>#<filter1>,<filter2>
```

Description

The testvariants command tests each of the variants listed by the listvariants (beta) command against a set of genomes, to determine which variants are found in which genomes. testvariants annotates each variation in the listvariants file with a flag for each allele of each tested genome to indicate if the variant is present in that allele.

Note that when comparing only two to three genomes, we recommend using calldiff instead of testvariants, as calldiff is less sensitive to the canonical alignment.

For more information on the algorithm that testvariants uses to compare variations, see "testvariants Algorithm" in the Appendix.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables testvariants (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.
`--input <listvariants_output>`	The listvariants file containing the variants to test for.
`--output <output_file>`	The output file. If this option is omitted, results are sent to `STDOUT`.
`--variants <variant_file> #<filter1>,<filter2>`	This option can be used once for each input variant file. Files can be ***var*** or ***masterVar*** files. Alternatively, you can read variant file names from a text file. The files must be specified with full path names, separated by spaces, and listed on a single line. For example, to read a list of files from dir_output.txt as the argument for the `--variants` option: `--variants \`cat dir_output.txt\`` Note that the argument is enclosed with backticks rather than single quotes. You can specify optional filters to turn selected calls into no-calls, using the syntax described for the varfilter (beta) tool. The filter syntax must be specified after each file name.

Input Files

testvariants takes as input a reference CRR file of the appropriate build, a set of two or more Complete Genomics variations files (***var-[ASM-ID].tsv.bz2***) or masterVar file (***masterVar-[ASM-ID].tsv.bz2***), and a listvariants output file generated based on the set of variation files to be used in the multi-genome comparison. These variant files may be in uncompressed or compressed (.bz2) form. In addition, you can filter the input files to turn selected calls into no-calls, as described for varfilter (beta).

Output Files

testvariants takes a listvariants output file containing the list of variations and appends one additional column for each genome variation file specified as input. Columns created by testvariants are headed with the ASM-ID, and contain one comparison flag for each allele in the tested genome. Table 12 lists the comparison flags.

Table 12: testvariants Flags: possible flags for each allele

Flag	Description
0	The variant is not present.
1	The variant is present.
N	The genome is no-called at this position.

Figure 22 shows testvariants output when the variations shown in Figure 21 are compared to variation calls in three genomes, ASM1, ASM2, and ASM3. In this example, ASM1 and ASM3 are homozygous for variant 1034, but ASM2 is heterozygous. ASM1 does not have variant 1036, ASM2 is heterozygous for the variant, and ASM3 is no-called at that position.

Note that each line in the testvariants output refers to a specific allele, not a genomic location. For example, variant 1038 is a 5 bp substitution that partially overlaps variant 1039 (a SNP), and variant 1040 (a deletion).

Table 13 describes the columns used in the example.

Figure 22: testvariants Output against Three Genomes

variantId	chromosome	begin	end	varType	reference	alleleSeq	xRef	ASM1	ASM2	ASM3
1034	chr1	972803	972804	snp	T	C	dbsnp:rs3128102	11	1	11
1035	chr1	972856	972857	snp	T	C	dbsnp:rs10267	11	11	11
1036	chr1	975024	975025	snp	G	T		0	1	NN
1037	chr1	975128	975129	snp	C	T	dbsnp:rs2275813	0	1	0
1038	chr1	975311	975316	sub	GGGGG	AAAAA		11	11	NN
1039	chr1	975311	975312	snp	G	A		1	0	NN
1040	chr1	975313	975316	del	GGG			0	1	NN
1041	chr1	975900	975901	snp	G	A		1N	0	1

Table 13: Column Descriptions for testvariants Output

	Column Name	Description
1	variantId	Sequential ID assigned to each variant.
2	chromosome	The chromosome of the variant.
3	begin	Zero-based reference offset of the beginning of the variant.
4	end	Zero-based reference offset of the end of the variant.
5	varType	The *varType* as extracted from the variant file.
6	reference	The reference sequence.
7	alleleSeq	The variant allele sequence as extracted from the variant file.
8	xRef	The *xRef* as extracted from the variant file.
9+	GSXXXXX-XXXX-XX-ASM	Column name corresponds to the sample ASM-ID. Contains one comparison flag for each allele in the tested genome. Comparison flags are described in Table 12.

Examples

In this example, we determine distribution of variations in the genomes of the Yoruban family trio, represented by the genomes NA19238, NA19239, and NA19240. This command takes the listvariants-produced list of all variations present in these genomes (`listvariants_YRI_trio.tsv`) and produces a file called `testvariants_YRI_trio.tsv` that lists the variant genotype for each family member. Note that in this example, the genomic variations provided to the listvariants and testvariants commands are identical.

```
cgatools testvariants \
--beta \
--reference /complete/build37.crr \
--input /complete/listvariants_YRI_trio.tsv \
--output /complete/testvariants_YRI_trio.tsv \
--variants \
/complete/GS19238-1100-37-ASM/GS00028-DNA_A01/ASM/var-GS19238-1100-37-ASM.tsv.bz2 \
/complete/GS19239-1100-37-ASM/GS00028-DNA_B01/ASM/var-GS19239-1100-37-ASM.tsv.bz2 \
/complete/GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/var-GS19240-1100-37-ASM.tsv.bz2
```

junctiondiff (beta)

Identifies junctions present in one genome that are absent from another genome B. The following sections describe its behavior:

Synopsis

```
cgatools junctiondiff --help
       --beta
       --reference <crr_file>
       --junctionsA <junction_fileA>
       --junctionsB <junction_fileB>
       --scoreThresholdA <thresholdA>
       --scoreThresholdB <thresholdB>
       --distance <arg>
       --minlength <arg>
       --output-prefix <prefix>
       --statout
```

Description

The junctiondiff tool identifies junctions present in one genome (genome A) that are absent from another (genome B).

Two junctions are considered equivalent if:

- They come from different files.
- The left and right positions of one junction are not more than 200 bp bases apart from the corresponding positions of another junction (the distance can be set by the user).
- The number of discordant mate pair alignments (i.e., the number of overlapping DNBs) supporting each junction is greater than or equal to the specified `--scoreThreshold` option value.
- They are on the same strands.

For more information, see "junctiondiff Algorithm" in the Appendix.

Command Line Options

Option	Description
`-h or --help`	Print command-line help.
`--beta`	Enables junctiondiff (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.
`-a <junction_fileA>` or `--junctionsA <junction_fileA>`	Input junction file for genome A.
`-b <junction_fileB>` or `--junctionsB <junction_fileB>`	Input junction file for genome B.
`-A <thresholdA>` or `--scoreThresholdA <thresholdA>`	The minimum number of discordant mate pair alignments (i.e., the number of overlapping reads) supporting the junction from genome A. If omitted, the minimum number defaults to 10.

Option	Description
`-B <thresholdA>` or `--scoreThresholdB <thresholdB>`	The minimum number of discordant mate pair alignments (i.e., the number of overlapping reads) supporting the junction from genome B. If omitted, the minimum number defaults to 0.
`-d` or `--distance <arg>`	Maximum distance between coordinates of potentially compatible junctions. If this option is omitted, the maximum distance is 200.
`-l` or `--minlength <arg>`	Minimum deletion junction length to be included into the difference file. If this option is omitted, the minimum length defaults to 500.
`-o <prefix>` or `--output-prefix <prefix>`	The path prefix for all output reports. This prefix can be used in two ways: ▪ If a path is specified (for example "`/home/myFiles`"), report files are saved to that location. ▪ If a string is specified (for example "`Run20111011`") it is appended to the start of the filename, and the file will be saved in the active directory.
`-S` or `--statout`	(Debug) Report various input file statistics. Experimental feature.

Input Files

junctiondiff takes as input a reference CRR file of the appropriate build and two junctions files for comparison. It accepts both the ***allJunctionsBeta*** and ***highConfidenceJunctionsBeta*** files that are generated from the Complete Genomics Structural Variation pipeline.

Output Files

junctiondiff creates an output file named `diff-input<file_nameA>`. This file contains the a list of the junctions from input file A that are not present in input file B. The output format is the same as the standard junctions file format, used in both the ***allJunctionsBeta*** and ***highConfidenceJunctionsBeta*** files.

If the `--statout` flag is used, junctiondiff generates a brief summary report (***report.tsv***).

Examples

Basic junctiondiff Operation

This command produces the junctions in the Yoruban NA19240 genome that are not present in her mother, NA19238.

```
cgatools junctiondiff \
--beta \
--reference /complete/build37.crr \
-a /complete/GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/allJunctionsBeta-GS19240-1100-37-ASM.tsv
-b /complete/GS19238-1100-37-ASM/GS00028-DNA_A01/ASM/allJunctionsBeta-GS19238-1100-37-ASM.tsv
```

junctiondiff Applied Iteratively to Multiple Genomes

junctiondiff can be run iteratively. The following command uses the output from the previous example as the input junction file A, and produces a list of the junctions in the Yoruban NA19240 genome that are not present in her mother, NA19238 or father, NA19239.

```
cgatools junctiondiff \
--beta \
--reference /complete/build37.crr \
-a /complete/diff-allJunctionsBeta-GS19240-1100-37-ASM.tsv \
-b /complete/GS19239-1100-37-ASM/GS00028-DNA_B01/ASM/var-GS19239-1100-37-ASM.tsv.bz2
```

SAM Conversion Tools

CGA Tools provides SAM-format conversion tools to facilitate downstream analysis and visualization, for example using the IGV genome browser.

Complete Genomics reads are initially mapped to the reference genome using a fast algorithm, and these initial mappings are later expanded and refined by local *de novo* assembly applied to putatively variant regions of the genome. The local *de novo* assembly identifies the most likely alleles for a variation interval (small region of the genome, less than 200 bases), and is followed by an optimization process that refines the allele choices. The mappings generated by the local *de novo* assembly and optimization process provide support for the called variations. Complete Genomics reads and their initial mappings to the reference genome are located in the MAP folder. The mappings generated by the local *de novo* assembly process (and their associated reads) are located in the EVIDENCE folder.

CGA Tools evidence2sam (beta) converts the additional mappings generated by local *de novo* assembly to SAM format. For pipelines that require reads and mappings in BAM format, the output of evidence2sam can be sent to standard output, and processed by SAM Tools.

Understanding Complete Genomics read structure and the differences between the initial mappings and evidence mappings is critical for proper use of evidence2sam. For more information, consult the Data File Formats document.

Complete Genomics calculates mapping quality in a very different manner to most other mapping tools. Some variant calling tools that take SAM/BAM files as input will expect the mapping quality scores to be different to those present for Complete Genomics reads and mappings, and may not call variants with the same accuracy as they would from native SAM/BAM files.

evidence2sam (beta)

Converts Complete Genomics evidence mappings to the SAM format. The following sections describe its behavior:

Synopsis

```
cgatools evidence2sam --help
       --beta
       --evidence-dnbs <evidence_file>
       –-output <sam_file>
       --reference <crr_file>
       --extract-genomic-region <arg>
       --keep-duplicates
       --add-allele-id
       --skip-not-mapped
       --add-mate-sequence
       --mate-sv-candidates
       --add-unmapped-mate-info
       --primary-mappings-only
       --consistent-mapping-range <arg>
```

Description

The evidence2sam tool converts Complete Genomics evidence mappings (located in the EVIDENCE folder) to the SAM format. For pipelines that require reads and mappings in BAM format, the output of evidence2sam can be sent to standard output, and processed by SAM Tools.

Important: evidence2sam does not convert the initial mappings (located in the MAP folder) to the SAM format.

Command Line Options

Option	Description
`-h` or `--help`	Prints command-line help.
`--beta`	Enables evidence2sam (currently beta-level).

`-e <evidence_file>`
or
`--evidence-dnbs <evidence_file>`
	Full or relative path to the ***evidenceDnbs*** file (***evidenceDnbs-[ASM-ID].tsv.bz2***).

`-o <sam_file>`
or
`--output <sam_file>`
	Specifies the full or relative path to the output SAM file. If this option is omitted, results are sent to `STDOUT`.

`-s <crr_file>`
or
`--reference <crr_file>`
	Specifies the reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.

`-r <arg>`
or
`--extract-genomic-region <arg>`
	Indicates the genomic coordinates that are converted to SAM, to avoid converting the entire file. Specify the region as a half-open interval `chr,from,to`. For example:
	`--extract-genomic-region chrX,15203639,15412498`

Option	Description
`--keep-duplicates`	Keeps local duplicates of DNB mappings. All the output SAM records will be marked as “not primary” if this option is used.
`--add-allele-id`	Generates interval ID (ZI:I) and allele ID (ZA:I) tags.
`--skip-not-mapped`	Does not output reads with no mappings.
`--add-mate-sequence`	Generates mate sequence (R2) and score (Q2) tags.
`--mate-sv-candidates`	Enables mating unique single arm mappings in SAM. Inconsistent mappings are normally converted as single arm mappings with no mate information provided. If this option is used, evidence2sam will mate unique single arm mappings in SAM including those on different stands and chromosomes. The tag "XS:i:1" is used to distinguish these "artificially" mated records. The MAPQ provided for these records is a single arm mapping weight
`--add-unmapped-mate-info`	Generates R2 and Q2 tags (as does `--add-mate-sequence`), but is applied to inconsistent mappings only.
`--primary-mappings-only`	Reports only the highest probability mapping for each read.
`--consistent-mapping-range <arg>`	Limits the maximum distance between consistent mates. If this flag is not provided, the default value is set to 1300 bp.

Input Files

evidence2sam takes as input the chromosome specific evidence DNB files, located in the ASM/EVIDENCE directory and a reference CRR file of the appropriate build.

Output Files

The evidence2sam converter takes one evidence mapping file (***evidenceDnbs-[ASM-ID].tsv.bz2***) as input and generates one SAM file as output. Each evidence mapping record from the input file is converted into a pair of corresponding SAM records, one record for each mate of a mate-pair read. Negative gaps in Complete Genomics mappings are removed from the reads and stored separately using GS/GQ/GC tags.

If evidence2sam is run with the `--add-allele-id` option, the optional ZA:I and ZI:I tags are used to store the allele number and evidence interval ID from the ***evidenceDnbs*** file, respectively. When run with the `--add-mate-sequence` flag, evidence2sam produces the SAM Tools R2 and Q2 tags for mate sequence and quality scores.

By default, evidence2sam de-duplicates duplicated DNB mappings, selecting the “best” mapping. Using the `--keep-duplicates` flag overrides this behavior, and causes all output SAM records to be marked as “not primary”.

For more information on the SAM output format, CIGAR string information, and the algorithm used to de-duplicate and select primary mappings, see “Representation of the Complete Genomics Data in SAM Output Format” in the Appendix.

Examples

This command pipeline uses evidence2sam to convert one ***evidenceDnbs*** file to SAM format. The output is then piped directly to SAM Tools to create an indexed, reference-sorted BAM file. This pipeline requires that SAM Tools be installed on your system and available from the location where this command is run. Note that the filename “result” (underlined below) is an arbitrary name of your choosing. However, it does need to be consistent between the SAM Tools sort and index commands.

```
cgatools evidence2sam \
--beta \
-e /GS19240-1100-36-ASM/GS0028-DNA_C01/ASM/EVIDENCE/evidenceDnbs-chr10-GS19240-1100-36-ASM.tsv.bz2 \
--reference /complete/build37.crr | \
samtools view -uS - | \
samtools sort - result && samtools index result.bam
```

VCF Conversion Tool

mkvcf (beta)

Translates small variant, copy number variation (CNV), structural variation (SV), and/or mobile element insertion (MEI) calls and annotations from one or more Complete Genomics genome assemblies into a single VCF-formatted file. We recommend that you run this tool for multi-genome comparisons of variations called using Complete Genomics Analysis Pipeline 2.0 and later versions. mkvcf is used to generate the ***vcfBeta*** file that is delivered with Complete Genomics data starting in Analysis Pipeline 2.2.

The following sections describe its behavior:

Synopsis

```
cgatools mkvcf --help
       --beta
       --reference <crr_file>
       --output <output_file>
       --field-names <list>
       --source-names <list>
       --genome-root <directory>
       --master-var <masterVar file>
       --include-no-calls
       --calibration-root <arg>
       --junction-file <junctions_file>
       --junction-score-threshold <arg>
       --junction-side-length-threshold <arg>
       --junction-distance-tolerance <arg>
       --junction-length-threshold
       --junction-normal-priority
       --junction-tumor-hc
```

Description

The mkvcf tool translates variant calls, including annotations and scores from one or more Complete Genomics genome assemblies, to a single VCF-formatted file. Thus mkvcf can be used for multigenome comparisons of variant calls. The number of genomes that can be compared and the data sources required to run mkvcf vary by variant type, as described in "Input Files."

Complete Genomics uses a variety of proprietary tags in the VCF file to fully capture the richness of our variant calls, annotations, and scores. These are fully described in "Field Tags." For more information on the algorithm that mkvcf uses to compare variations, see "mkvcf Translation Details" in the appendix.

Analysis Pipeline Version Effects

mkvcf is fully compatible with data generated using Complete Genomics Analysis Pipeline version 2.0 and later. mkvcf has limited backwards compatibility when generating single-genome VCFs. This varies by variant type and is summarized in "Input Files." mkvcf does not support multi-genome comparison of genomes analyzed on Analysis Pipelines pre-2.0.

In CGA Tools version1.7, mkvcf output for paired-sample LAF measurements (produced by the Cancer Sequencing Service) now uses the CGA_LAFP, CGA_ULAFP, and CGA_LLAFP tags, as introduced in Analysis Pipeline version 2.4.

Data Structure Requirement

Use of the `mkvcf --genome-root` flag requires input files to be located in the directory hierarchy in which the data package was originally delivered by Complete Genomics.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables mkvcf (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file.
`--output <output_file>`	Full or relative path to the output file. If this option is omitted, results are sent to `STDOUT`.
`--field-names <list>`	Comma-separated list of format fields to include in the output. If this option is omitted, all fields are included. For a description of each field, see "Field Tags." Available fields, organized by source type, are: ▪ masterVar-related fields: GT,PS,NS,AN,AC,SS,FT,CGA_XR,CGA_FI,GQ,HQ,EHQ,CGA_CEHQ, GL,CGA_CEGL,DP,AD,CGA_RDP,CGA_ODP,CGA_OAD,CGA_ORDP, CGA_PFAM,CGA_MIRB,CGA_RPT,CGA_SDO,CGA_SOMC, CGA_SOMR,CGA_SOMS,CGA_SOMF,AF,CGA_ALTCALLS ▪ CNV-related fields: GT,CGA_GP,CGA_NP,CGA_CP,CGA_PS,CGA_CT,CGA_TS,CGA_CL, CGA_LS,CGA_SCL,CGA_SLS,CGA_LAFS,CGA_LLAFS,CGA_ULAFS,C GA_LAFP,CGA_LLAFP,CGA_ULAFP ▪ MEI-related fields: GT,FT,CGA_IS,CGA_IDC,CGA_IDCL,CGA_IDCR,CGA_RDC, CGA_NBET,CGA_ETS,CGA_KES ▪ SV-related fields: GT,FT,CGA_BF,CGA_MEDEL,MATEID,SVTYPE,CGA_BNDG, CGA_BNDGO,CGA_BNDMPC,CGA_BNDPOS,CGA_BNDDEF, CGA_BNDP SV-ALL may be specified as shorthand for all SV-related fields.
`--source-names <list>`	Comma-separated list of variant source names. If this option is omitted, all source names are included. The following source names are available: ▪ masterVar: Includes records from the ***masterVar*** file. ▪ CNV: Includes CNV-related records. ▪ SV: Includes records derived from junctions files. ▪ MEI: Includes records describing mobile element insertions. Some of these source types are only available for more recent pipeline versions, and some of these source types do not support multi-genome VCFs. For more information about which source types are available for which versions of the Complete Genomics pipeline software, see "Input Files."

Option	Description
`--genome-root <directory>`	The genome root directory that contains the ASM/REF and ASM/EVIDENCE subdirectories. For example: `/data/GS00118-DNA_A01` You must supply this option for each genome in the VCF, unless you are using `--source-names=masterVar` and you have specified the `--master-var` option for each genome in the VCF, or you are using `--source-names=SV` and have specified `--junction-file` for each input genome.
`--master-var <masterVar file>`	For each genome to include in the VCF, the ***masterVar*** file. If `--genome-root` parameter is given, this parameter defaults to the ***masterVar*** in the given genome-root. If the number of genomes specified via `--genome-root` is non-zero, then the number of ***masterVar*** specifications must be either 0 or the same as the number of genomes. The order of ***masterVar*** files should correspond to that of genome roots.
`--include-no-calls`	Include small variants VCF records for loci that are no-called across all input genomes. Note that no-calls in small variant loci with at least one called allele across input genomes are always output.
`--calibration-root <arg>`	The directory containing score calibration data. The directory should contain directories `version0.0.0` and `version2.0.0`. For example: `/home/complete/var-calibration-v1` This option is only required if CGA_CEHQ or CGA_CEGL are included in the `--field-names` parameter.
`--junction-file <junctions_file>`	For each genome to include in the VCF, the junctions file. If `--genome-root` parameter is given, this parameter defaults to the junctions file in the given genome root. If the number of genomes specified via `--genome-root` is non-zero, then the number of junction files must be either 0 or the same as the number of genomes. The order of junction files should correspond to that of genome roots.
`--junction-score-threshold <arg>`	Minimum number of discordant mate pairs for a junction that is required to be labeled as PASS in the FT record. If this option is omitted, the default is 10.
`--junction-side-length-threshold <arg>`	Minimum "junction side length" for a reported junction that is required to be labeled as PASS in the FT record. If this option is omitted, the default is 70 bp.
`--junction-distance-tolerance <arg>`	Maximum allowed distance between junctions considered to match (i.e. potentially reflect the same evolutionary event). If this option is omitted, the default is 200 bp.
`--junction-length-threshold`	Minimum length between breakpoints required to call an intrachromosomal junction. If this option is omitted, the minimum length defaults to 500 bp.

Option	Description
`--junction-normal-priority`	Normal junction priority for VCF output. Should be used only when comparing two genomes, a tumor and its matched normal. If this switch is present, then each cluster of compatible junctions will be represented by the junction from the normal genome (the first one in the list of genomes). Otherwise it might represented by either normal or tumor junction.
`--junction-tumor-hc`	Output only high confidence junctions from the second of two genomes. Useful as a means of identifying a set of high-confidence somatic junctions.

Input Files

Table 14 lists the input files required to convert each class of variant calls, the version compatiblity for single- and multi-genome conversion, and constraints on how many genomes can be processed at one time. Files may be specified implicitly, via the `--genome-root` parameter, or in some cases explicitly, via the `--master-var` and/or `--junction-file` parameters.

Table 14: Variant Comparison Constraints

Variant Type	Number of Genomes That Can Be Specified	Required Input File Types	Analysis Pipeline Compatibility: Single-Genome VCF Conversion	Analysis Pipeline Compatibility: Multi-Genome VCF Conversion
MEI	One genome	***mobileElementInsertionsBeta****	Version 1.12.0 and later	Version 2.0.0 and later
SV	Two genomes	***allJunctions**** or ***highConfidenceJunctions****	Version 1.10.0 and later	Version 2.0.0 and later
CNV	Any number of genomes	***cnvDetails**** and/or ***somaticCnvDetails****	Version 2.0.0 and later	Version 2.0.0 and later
Small Variants	Any number of genomes	***masterVar**** or ***var****	Version 1.12.0 and later. Also compatible with ***masterVar*** files generated from pre-1.12 data using generatemasterVar (beta)	Version 2.0.0 and later

Output Files

mkvcf produces a single file in the VCF 4.1 format, regardless of number of genomes in the input. If `--output` is used to provide a file name, output will be written to that file; otherwise, output will be written to STDOUT.

Running mkvcf without specifying specific field names (via the --field-names flag) results in mkvcf including all tags by default. For small variants, this includes tags such as CGA_CEGL, that provide calibrated scores (for more information see *Complete Genomics Small Variant Score Calibration Methods*). To properly compute the calibrated scores, mkvcf needs access to version 2 of the calibrated score files.

Examples

Produce a single genome VCF, including small variants, CNVs, SVs, and MEIs

Suppose you want to convert all types of variant calls for a single genome into a VCF file, where the top-level directory for the genome is `/data/GS00118-DNA_A01`. You can do this as follows:

```
cgatools mkvcf \
--beta \
--genome-root /data/GS00118-DNA_A01 \
--source-names masterVar,SV,MEI,CNV \
--reference /home/complete/data/ref/build37.crr \
--output GS00118-DNA_A01.vcf
```

Produce a two-genome VCF including small variants, CNVs, and SVs

Suppose you want to create a VCF for a tumor/normal comparison that includes small variants, CNVs, and SVs. The top-level directories for the genomes are each in the `/data` directory. You can do this as follows:

```
cgatools mkvcf \
--beta \
--genome-root /data/GS00118-DNA_A01 \
--genome-root /data/GS00122-DNA_D06 \
--source-names masterVar,SV,CNV \
--reference /home/complete/data/ref/build37.crr \
--output tumor_normal.vcf
```

Note that where the goal is the identification of somatic changes between a tumor and a normal, the baseline genome (typically, normal) for the identification of somatic changes should be specified first and the derived/non-baseline genome (typically, tumor) should be specified second. This ensures correct behavior of the `--junction-normal-priority` and `--junction-tumor-hc` flags.

Produce a four-genome VCF including small variants and CNV

Suppose you want to create a VCF for four separate genomes that includes small variants and CNVs. The top-level directories for the genomes are each in the `/data` directory. You can do this as follows:

```
cgatools mkvcf \
--beta \
--genome-root /data/GS00118-DNA_A01 \
--genome-root /data/GS00122-DNA_B03 \
--genome-root /data/GS00122-DNA_D05 \
--genome-root /data/GS00122-DNA_E02 \
--source-names masterVar,CNV \
--reference /home/complete/data/ref/build37.crr \
--output four_genome.vcf
```

Field Tags

Information about each variant type is sourced from different files (see `--source-names` parameter description) and described using different field tags. Most tags apply to only one source, but a few apply to two or more. The field tags output by mkvcf are described in the ***somaticVcfBeta*** file section of the Complete Genomics Cancer Sequencing Service Data File Formats document.

Master Variation File Format Conversion Tool

generatemasterVar (beta)

Produces an integrated master variation file to report the variant calls and annotation information produced by the Complete Genomics assembly process. Note that a ***masterVar*** file is delivered with Complete Genomics data after the Assembly Software version 1.12.

The following sections describe its behavior:

Synopsis

```
cgatools generatemastervar --help
       --beta
       --reference <crr_file>
       --output <mastervar_file>
       --variants <variant_file>
       --annotations <annotation_list>
       --genome-root <directory>
       --repmask-data <repeatmask_data>
       --segdup-data <seg_dup_file>
```

Description

The generatemasterVar tool produces an integrated master variation file (***masterVarBeta***) to report the variant calls and annotation information produced by the Complete Genomics assembly process. Unlike the ***var*** file, the 2 alleles for each locus are reported as one record, simplifying data processing. With the release of Complete Genomics assembly software version 1.12, the ***masterVarBeta-[ASM-ID].tsv.bz2*** file became a standard deliverable, located in the ASM subfolder of each genome assembly. The generatemasterVar tool included in CGA Tools enables the generation of ***masterVarBeta*** from genomes assembled using earlier releases of the assembly software. The file format is derived heavily from the existing variation file format and can be used with all CGA Tools commands anywhere a variation file is expected. A detailed description of the ***masterVarBeta*** file format can be found in Complete Genomics Data File Formats documents.

Analysis Pipeline Version Effects

The ***masterVarBeta*** files produced by Analysis Pipeline version 2.4 include new information including single-sample LAF measurements, allele frequencies reported in dbSNP, ambiguous calls, a new somatic score (*fisherSomatic*), and changes to the variant flagging system. generatemastervar in CGA Tools version 1.7 will output files using the format adopted in Analysis Pipeline version 2.4, irrespective of the original analysis pipeline used to analyze the input genomes. For example, variant quality filters will be provided in *alleleXVarFilter* columns instead of the *varQuality* or *somaticQuality* columns. The output files will also contain new columns introduced in Analysis Pipeline version 2.4 (such as *alleleXFreq* and *alleleXAlternativeCalls*) though these columns will be empty if the input files were analyzed on Analysis Pipelines pre-2.4.

generatemastervar can output the *fisherSomatic* score (a score introduced in Analysis Pipeline version 2.4 that measures confidence in called somatic variants, complementing *somaticScore*) and the

FET30 flag (which indicates lower confidence in the somatic call). These cancer-related features require the source genome to have been sequence using the Complete Genomics Cancer Sequencing Service.

Analysis Pipeline version 2.0 includes both diploid and non-diploid model CNV calls. generatemasterVar can process either or both types of calls if this data is available as specified using the CNV parameter of the `--annotations` option. For data generated with Analysis Pipeline versions 1.10, 1.11, and 1.12, generatemasterVar processes a single value: you specify either diploid model (normal CNV processing) CNV calls or non-diploid model (tumor CNV processing) using the CNV Diploid and CNV Nondiploid parameters of the `--annotations` option. If neither diploid nor non-diploid model data is available, such as is the case with the Analysis Pipeline version 1.8 and earlier, and any of the CNV parameters is specified, generatemasterVar will produce an error.

Data Structure Requirement

The generatemastervar tool requires input files to be located in the directory hierarchy in which the data package was originally delivered by Complete Genomics.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables generatemasterVar (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.
`--output <mastervar_file>`	Full or relative path to the output masterVar file. If this option is omitted, results are sent to `STDOUT`.
`--variants <variant_file>`	Full or relative path to the input variant file.
`--annotations <annotation_list>`	Comma-separated list of annotations to add to each line. Possible annotation values are: `copy`, `evidence`, `gene`, `ncrna`, `repeat`, `segdup`, `cnv`, `cnvDiploid`, or `cnvNondiploid`. See Table 15 for more information.
`--genome-root <directory>`	The genome directory that contains an intact ASM subdirectory. For example: `/data/GS00118-DNA_A01`.
`--repmask-data <repeatmask_data>`	Full or relative path to the file that contains repeat masker data.
`--segdup-data <seg_dup_file>`	Full or relative path to the file that contains segmental duplications data.

Input Files

generatemasterVar merges the variant calls in the ***var*** file with annotations from a variety of sources. Table 15 lists all required and optional generatemasterVar annotation source files, and provides information on where certain annotation source files can be obtained. These files do not need to be explicitly specified but are inferred from the list of annotations in the `--annotations` option.

Table 15: Variant and Annotation Data Sources

Source	Description
var* or ***masterVarBeta****	Source of variant calls, and external reference (*xRef*) annotations. Data is copied from the input file. If the input file is the variation file (***var-[ASM-ID].tsv.bz2***), the *xRef* data is re-formatted to a semicolon-delimited list of all xRef annotations for all alleles of the locus. If the input file is the master variations file (***masterVar-[ASM-ID].tsv.bz2***), its annotation columns are simply copied to the output.
evidence	Data from Complete Genomics evidence files.
gene	Data from the Complete Genomics gene annotation file (***gene-[ASM-ID].tsv.bz2***).

Source	Description
ncRNA	Data from the Complete Genomics non-coding RNA annotation file (***ncRNA-[ASM-ID].tsv.bz2***).
repeat	RepeatMasker information. Specify the RepeatMasker annotation data file with the `--repmask-data` option. The data is derived from the RepeatMasker table available from the UCSC genome browser website or on Complete Genomics' FTP site. See "Obtaining Ancillary Files for Use with CGA Tools" for download instructions.
segdup	Information about segmental duplications. Specify the segmental duplication data file with the `--segdup-data` option. The data is derived from the Segmental Duplications table available from the UCSC genome browser website or on Complete Genomics' FTP site. See "Obtaining Ancillary Files for Use with CGA Tools" for download instructions.
cnv	Information about the CNV calls made by the Complete Genomics pipeline for the region that covers this locus. If the genome package was generated by the Analysis Pipeline version 2.0, both the diploid model and non-diploid model CNV calls will be added as columns. Note: including both "cnv" and other cnv sources (such as cnvDiploid) on the command line will result in duplicate columns. If the genome package was generated by Analysis Pipeline versions 1.10, 1.11, and 1.12, either diploid model (normal CNV processing) CNV calls or non-diploid model (tumor CNV processing) CNV calls are added when CNV is specified. For these pipeline releases, only one of the two CNV values is produced. If neither is available, such as is the case with the Analysis Pipeline version 1.8 and earlier, generatemasterVar will produce an error.
cnvDiploid	Information about the diploid model CNV calls (*calledPloidy* and *relativeCoverageDiploid*) made by the Complete Genomics pipeline for the region that covers this locus. Note that this information is only available in genome packages generated by the Analysis Pipeline version 2.0 or later.
cnvNondiploid	Information about the non-diploid model CNV calls (*calledLevel*, *relativeCoverageNondiploid,* and, if available, *bestLAFsingle, lowLAFsingle,* and *highLAFsingle)* made by the Complete Genomics pipeline for the region that covers this locus. Note that this annotation option is only available in genome packages generated by the Analysis Pipeline version 2.0 or later. *LAFsingle*-related information is only available for version 2.4 or later.
cnvSomNondiploid	Information about the non-diploid somatic CNV calls (*somaticCalledLevel*, *relativeCoverageSomaticNondiploid, bestLAFpaired, lowLAFpaired,* and *highLAFpaired*) made by the Complete Genomics pipeline for the region that covers this locus. Note that this option is only available in genome packages produced with the Cancer Sequencing Service.
fisherSomatic	Provides the *fisherSomatic* score – a score measuring the confidence in called somatic variants introduced in Analysis Pipeline version 2.4. The score is computed using a one-tailed Fisher's Exact Test on counts of reads supporting alt and reference alleles in the baseline (usually matched normal) and non-baseline (usually tumor) samples. The given score is intended to have a Phred-like interpretation of -10*log10(probability of an erroneous call), though details of the count tabulation make the intended calibration only approximate. Note that this option is only available in genome packages produced with the Cancer Sequencing Service.

Output Files

A detailed description of the ***masterVar*** file format that is created by generatemasterVar, including example output, can be found in the Complete Genomics Data File Formats document.

The ***masterVar*** file format is convenient for manipulating Complete Genomics data with tools outside the CGA Tools package. For the ***masterVar*** file to maintain compatibility with CGA Tools, the following rules must be followed:

1. Header line TYPE must be preserved as is. It is recommended, but not required, that header values are also preserved.
2. Every locus data line must contain the same set of columns. The column header line (starting with a ">" character) must be present and must contain the same number of columns as the rest of the file.
3. Order of lines must remain intact. The loci in the file are sorted in the order of the reference.
4. The values of the mandatory columns must not be modified and columns themselves must not be removed. In particular, loci may not be split or merged.

Example

Here we generate ***masterVar*** from the NA19240 ***var*** file.

```
cgatools generatemastervar \
--beta \
--reference /ref/build37.crr \
--output mastervar-GS19240-1100-37-ASM.tsv \
--variants /GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/var-GS19240-1100-37-ASM.tsv.bz2 \
--annotations copy,evidence,gene,ncrna,repeat,segdup,cnv \
--genome-root /GS19240-1100-37-ASM/GS00028-DNA_C01 \
--repmask-data /ref/rmsk37.tsv.gz \
--segdup-data /ref/segdup37.tsv.gz
```

Filtering and Annotation Tools

Most files used as input or output for CGA Tools are simple tab-delimited files that can be interpreted as tables. As such, CGA Tools provides tools that manipulate the files as tables.

- varfilter (beta)
- join (beta)
- junctions2events (beta)

varfilter (beta)

Filters the content of ***var*** or ***masterVarBeta*** files based on one or more call selectors. The following sections describe its behavior:

Synopsis

```
cgatools varfilter --help
       --beta
       --reference <crr_file>
       --input <variant_file>#<filter1>,<filter2>
       --output <output_file>
```

Description

The varfilter command applies filters to turn selected calls in ***var*** or ***masterVarBeta*** files into no-calls. For example, filters can be applied to low quality calls in a ***var*** or ***masterVarBeta*** file to turn them into no-calls. Filtering variants before performing genome comparison analyses may be desirable to limit comparisons to only interesting or high quality variants.

Note that variant filtering of ***var*** or ***masterVarBeta*** input files is enabled for all CGA Tools that use these input files, such as calldiff, snpdiff, and listvariants/testvariants. Thus, you need not run the varfilter tool separately to generate a filtered input file for downstream analysis.

You construct a filter by listing one or more of the call selectors outlined in Table 16, separated by colons. The call selectors are the only fields available for filtering. You can create complex filtering criteria by applying more than one filter to the input file: specify multiple filters by concatenating individual filters into a comma-separated list.

To specify filters on the varfilter command line, append a "#" sign and the filter or filter list to the input file. For example, to produce a version of the ***var*** file containing only scored snp calls (turning everything that is not a snp into no-calls) you would use the following file and filter specification:

```
/path/to/var.tsv.bz2#varType!=snp
```

Table 16: varfilter Call Selectors

Call Selector	Description
`hom`	Selects scored calls in homozygous loci.
`het`	Selects scored calls that are not in homozygous loci.
`varType=XX`	Selects scored calls whose *varType* is XX, where XX can be one of `snp`, `ins`, `del`, `sub`, `ref`, `no-call-rc`, `no-call-ri`, `no-call`, `No-ref`, or `PAR-called-in-X`.
`varScoreVAF<XX`	Selects calls whose *varScoreVAF*<XX, where XX is a positive integer representing the confidence in the call.
`varScoreEAF<XX`	Selects calls whose *varScoreEAF*<XX, where XX is a positive integer representing the confidence in the call.
`varQuality!=VQHIGH`	Selects scored calls whose *varQuality* is not `VQHIGH` (that is, `VQLOW` and empty). Supports samples analyzed using Analysis Pipelines before version 2.4.

Command Line Options

Option	Description
`-h or --help`	Print command-line help.
`--beta`	Enables varfilter (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full or relative path to the reference file. You can use a positional argument to specify the CRR file.
`--input <filepath>#<filter1>[,<filter2>]`	Input file and filters to apply. Indicate the full or relative path to the input file followed by a pound sign (#) followed by a list of filters, separated by commas. Each filter can include more than one call selectors separated by colons. The call selectors that make up the filters are described in Table 16.
`--output <output_file>`	Full or relative path to the output file. If this option is omitted, results are sent to `STDOUT`.

Input Files

varfilter takes as input a Complete Genomics variant (***var-[ASM-ID].tsv.bz2***) or master variant file (***masterVarBeta-[ASM-ID].tsv.bz2***), and a reference CRR file of the appropriate build.

Output Files

varfilter produces a filtered version of the input files based on the user-specified filters.

Example

Here is an example of the argument for the `--input` option that filters out homozygous SNPs with *varScoreVAF* < 25 and heterozygous insertions with *varScoreEAF* < 50.

Syntax Note: Special characters in the command line must be escaped as shown in this example.

```
cgatools varfilter \
--beta \
--reference /home/complete/build37.crr \
--input /GS19240-1100-37-ASM/GS00028-DNA_C01/ASM/ \
var-GS19240-1100-37-ASM.tsv.bz2#hom:varType=snp:varScoreVAF\<25, \
het:varType=ins:varScoreEAF\<50 \
--output var-GS19240-1100-37-ASM_filtered.tsv
```

join (beta)

Merges the results of two delimited input files. The following sections describe its behavior:

Synopsis

```
cgatools join --help
       --beta
       --input <file1> <file2>
       --output <output_file>
       –-match <specification>
       --overlap <overlap_spec>
       --output-mode <arg>
       --overlap-mode <arg>
       --select <output_fields>
       --always-dump
       --overlap-fraction-A <fraction>
       --boundary-uncertainty-A <arg>
       --overlap-fraction-B <arg>
       --boundary-uncertainty-B <arg>
```

Description

The join tool works like a database join to merge the results of two delimited input files. It can be used, for example, to annotate the variant file with your own set of annotations.

By default, an output record is produced for each match found between file A and file B, but output format can be controlled by the `--output-mode` option.

The limitation of the join tool is that file B must fit into memory.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables join (currently beta-level).

`--input <fileA> <fileB>`
or
`--input <fileB>`
	Full or relative path names to the files used as input: there must be exactly two input files to join. If only one file is specified by name, file A is taken to be standard in and file B is the named file. File B is read fully into memory, and file A is streamed.
	Columns from file A appear first in the output.

`--output <output_file>`	Full or relative path to the output file. If this option is omitted, results are sent to `STDOUT`.
`--match <match_spec>`	A match specification: column name from file A followed by a colon, then column name from file B. For example: `--match begin:begin` Specify additional `--match` options on the command line to pair additional columns.

Option	Description
`--overlap <overlap_spec>`	Overlap specification: range definition for files A and B, separated by a colon. A range definition can be: ▪ Two columns (indicated by the column names), in which case they are interpreted as the beginning and end of the range. ▪ One column, in which case the range is defined as the 1-base range starting at the given value. The records from the two files must overlap in order to be considered for output. Two ranges are considered to overlap if the criteria specified by the `--overlap-mode` option are met. For additional specificity, use multiple overlap fields in the same command.

`-m <arg>`
or
`--output-mode <arg>`
	Output mode. The argument is one of the following:
	▪ `full`: Print an output record for each match found between file A and file B.
	▪ `compact`: Print at most one record for each record of file A, joining the file B values by a semicolon and suppressing repeated B values and empty B values.
	▪ `compact-pct`: Same as `compact`, but for each distinct B value, annotate with the percentage of the A record that is overlapped by B records with that B value. Percentage is rounded up to nearest integer.

`--overlap-mode <arg>`	Overlap mode. The argument is one of the following: ▪ `strict`: (Default) Range A and B overlap if A.begin < B.end and B.begin < A.end. ▪ `allow-abutting-points`: Range A and B overlap they meet the strict requirements, or if A.begin <= B.end and B.begin <= A.end and either A or B has zero length.
`--select <output_fields>`	Specifies the set of fields to select for output. Indicate the source file followed by a period followed by the field name for each field; separate multiple files with commas. For example: `a.locus,a.ploidy,a.varType,b.region`

`-a`
or
`--always-dump`
	Include every record of A in the output, even if there are no matches with file B.

`--overlap-fraction-A <fraction>`	Minimum fraction of A region overlap for filtering output. If this option is omitted, the A overlap fraction defaults to 0.
`--boundary-uncertainty-A <arg>`	Boundary uncertainty for overlap filtering. Specifically, records failing the following boundary uncertainty calculation are not included in the output: `overlap length >= overlap-fraction-A * (A-range-length - boundary-uncertainty-A)` If this option is omitted, the boundary uncertainty defaults to 0.
`--overlap-fraction-B <arg>`	Minimum fraction of B region overlap for filtering output. If this option is omitted, the B overlap fraction defaults to 0.
`--boundary-uncertainty-B <arg>`	Boundary uncertainty for overlap filtering. Specifically, records failing the following boundary-uncertainty calculation are not included in the output: `overlap length >= overlap-fraction-B * (B-range-length - boundary-uncertainty-B)` If this option is omitted, the boundary uncertainty defaults to 0.

Input Files

Two tab delimited text files with column-headers and a set of specifications necessary to determine overlapping regions between the two files.

Output Files

join Produces a tab-delimited file, based on the content of the two input files, where the overlap criteria and all specified join parameters have been applied to both input files.

Example

Annotate a Variant File

Suppose that you are interested in two regions of the genome and would like to extract the variants present in these regions from a Complete Genomics variant file.

For example, suppose you have the following file (file1.tsv):

Figure 23: Example join Input File (file1.tsv)

```
chromosome      begin         end         region
chr1            13            23          InterestingRegion1
chr2            19            20          InterestingRegion2
```

Figure 24: Variant File (var.tsv)

>locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink	xRef	alleleFreq	alternativeCalls
1	2	all	chr1	0	1	no-call	=	?							
2	2	all	chr1	1	7	ref	=	=							
3	2	1	chr1	7	8	snp	C	T	87	87	PASS	1	dbsnp:123		
3	2	2	chr1	7	8	ref	C	C	57	57	PASS	2	dbsnp:123		
4	2	all	chr1	8	13	ref	=	=							
5	2	1	chr1	13	13	ins		A	15	36	VQLOW				
5	2	2	chr1	13	13	ins		A	19	42	VQLOW				
6	2	all	chr1	13	22	ref	=	=							
7	2	1	chr1	22	24	del	AT		60	47	PASS	1			
7	2	2	chr1	22	24	ref	AT	AT	75	55	PASS	2			
8	2	all	chr1	24	29	ref	=	=							
9	2	1	chr1	29	31	ref	CC	CC	57	57	PASS	1			
9	2	2	chr1	29	31	no-call-ri	CC	TN	65	65	PASS	2			
10	2	all	chr1	31	40	ref	=	=							
11	2	1	chr1	40	41	ref	G	G	129	101	PASS	1			
11	2	1	chr1	41	41	ins		GG	118	120	PASS	1			
11	2	2	chr1	40	41	snp	G	T	479	479	PASS	2			
12	2	all	chr1	41	42	ref	=	=							
13	1	all	chr2	0	10	ref	=	=							
14	1	1	chr2	10	11	no-call-rc	C	N	50	47	PASS				
15	1	all	chr2	11	18	ref	=	=							
16	1	1	chr2	18	20	sub	TT	CG	102	102	PASS				
17	1	all	chr2	20	27	ref	=	=							

You can annotate the variant file from Figure 24 as follows:

```
cgatools join \
--beta \
--input var.tsv file1.tsv \
--overlap begin,end:begin,end \
--select 'a.*,b.region' \
--match chromosome:chromosome
```

The result is all the records of the variant file that overlapped with your regions of interest, sent to standard out (no `--output` option specified), as shown in Figure 23.

Figure 25: join Example Results

>locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	varScoreVAF	varScoreEAF	varFilter	hapLink	xRef	alleleFreq	alternativeCalls	region
6	2	all	chr1	13	22	ref	=	=								InterestingRegion1
7	2	1	chr1	22	24	del	AT		60	47	PASS	1				InterestingRegion1
7	2	2	chr1	22	24	ref	AT	AT	75	55	PASS	2				InterestingRegion1
16	1	1	chr2	18	20	sub	TT	CG	102	102	PASS					InterestingRegion2

To accomplish this, the join tool first reads the annotations file (file B) into memory. Then it streams the variant file (file A); for each record of file A, it finds the records of file B that match the user-selected columns or that overlap the record. As a consequence of this implementation, file B must fit into memory, but file A may be arbitrarily large. Additionally, the output records are in the same order as they are found in file A.

Annotate calldiff Output with Gene Information

Suppose that you have a SomaticOutput file from calldiff and would like to annotate it with gene information from a Complete Genomics gene file. Because you would like to match fields from multiple columns that are present in both files, you could use the following command to create a merged file called `somaticOutputWithGenes.tsv`.

```
cgatools join \
--input /somaticOutputFile.tsv \
--input /GSXXXXX-DNA_C01_1120/ASM/gene-GS0000XXXXX-ASM.tsv.bz2
–-match locus:locus \
--match allele:allele \
--match chromosome:chromosome \
--match varType:varType \
--match reference:reference \
--match begin:begin \
--match end:end \
–-select='a.*,b.*' \
--output somaticOutputWithGenes.tsv
```

junctions2events (beta)

Groups related junctions and annotates each group with information about the structural rearrangement ("event") that these junctions represent. The following sections describe its behavior:

Synopsis

```
cgatools junctions2events --help
       --beta
       --reference <crr_file>
       --output-prefix <prefix>
       --junctions <junctions_file>
       --all-junctions <alljunctions_file>
       --repmask-data <repeat_file>
       --gene-data <gene_file>
       --regulatory-region-length <arg>
       --contained-genes-max-range <arg>
       --max-related-junction-distance <arg>
       --max-pairing-distance <arg>
       --max-copy-target-length <arg>
       --max-simple-event-distance <arg>
       --mobile-element-names <arg>
       --max-distance-to-m-e <arg>
```

Description

Junctions are discontinuities in a sample genome relative to the reference genome. They occur when a sample genome contains contiguous sequence that is not adjacent and/or in the same orientation in the reference genome. Structural variations such as duplications, deletions, inversions, and translocations are represented by one or more junctions. The junctions2events tool groups related junctions and produces output in which related junctions are annotated with information about the structural rearrangement ("event") that these junctions represent.

For more information on how the junctions2events command operates, see "junctions2events Algorithm" in the Appendix.

Repeat Masker and gene data files necessary to run this command can be downloaded from the Complete Genomics site. See "Obtaining a Reference Human Genome for Use with CGA Tools" for download instructions.

Analysis Pipeline Version Effects

In Complete Genomics Analysis Pipeline version 2.0 and later, the junctions2events tool is run within the pipeline, and two files, ***allSvEventsBeta-[ASM-ID].tsv*** and ***highConfidenceSvEventsBeta-[ASM-ID].tsv*** files are provided as part of the a standard deliverable. These files are located in the SV subfolder within the ASM directory of each genome assembly. They are differentiated by the list of junctions used as input to the tool. The junctions2events tool included in CGA Tools enables the generation of ***allSvEventsBeta*** and ***highConfidenceSvEventsBeta*** from genomes assembled using earlier releases of the Analysis Pipeline.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--beta`	Enables junction2events (currently beta-level).
`--reference <crr_file>`	The reference CRR file. Specify the full path to the reference file. You can use a positional argument to specify the CRR file.
`--output-prefix <prefix>`	The path prefix for all output reports. This prefix can be used in two ways: ▪ If a path is specified (for example "`/home/myFiles`"), report files are saved to that location. ▪ If a string is specified (for example "`Run20111011`") it is appended to the start of the filename, and the file will be saved in the active directory.
`--junctions <junctions_file>`	Full or relative path to the primary input junction file.
`--all-junctions <alljunctions_file>`	Superset of the input junction file to use when searching for the related junctions. If this option is omitted, only the junctions in the primary junction file are used.
`--repmask-data <repeat_file>`	Full or relative path to the file that contains repeat masker data.
`--gene-data <gene_file>`	Full or relative path to the file that contains gene location data. Note that this is not the gene file delivered with the sample genome, but rather a reference assembly-specific gene annotation file that can be downloaded from the Complete Genomics website. See "Obtaining Ancillary Files for Use with CGA Tools".
`--regulatory-region-length <arg>`	Length of the region upstream of the gene that may contain regulatory sequence for the gene. Junctions that connect this region to another gene will be annotated as a special kind of gene fusion. If this option is omitted, the length used is `7500`.
`--contained-genes-max-range <arg>`	Maximum length of a copy or deletion event to annotate with all genes that overlap the copied or deleted segment. Negative value causes all events to be annotated regardless of the length. If this option is omitted, the maximum length is set to `-1`.
`--max-related-junction-distance <arg>`	Junctions occurring within this distance are presumed to be related. If this option is omitted, the distance defaults to `700`.
`--max-pairing-distance <arg>`	Maximum allowed distance between junction sides when searching for paired junctions caused by the same event. If this option is omitted, the distance defaults to `10000000`.

Option	Description
`--max-copy-target-length <arg>`	Pairs of junctions will be classified as a copy event only if the length of the implied copy target region is below this threshold. If this option is omitted, the threshold is set to `1000`.
`--max-simple-event-distance <arg>`	When given a choice of identifying an event as a mobile element copy or as a simple deletion/duplication, prefer the latter explanation if the length of the affected sequence if below this threshold. If this option is omitted, the threshold is set to `10000000`.
`--mobile-element-names <arg>`	Comma-separated list of the names of the mobile elements that are known to be active and sometimes copy flanking 3' sequence. Values can be `L1HS`, `AluY`, and `SVA`.
`--max-distance-to-m-e <arg>`	Maximum allowed distance from the junction side to the element when searching for a mobile element related to a junction. If this option is omitted, the maximum distance is set to 2000.
`--max-related-junction-output <arg>`	Maximum number of related junctions included in the *RelatedJunctionIds* field. If this option is omitted, the default is set to 100.

Input Files

junctions2events takes as input a reference CRR file of the appropriate build, a primary input junctions file that corresponds to either the ***allJunctionsBeta-[ASM-ID].tsv*** or ***highConfidenceJunctionsBeta-[ASM-ID].tsv***, and, optionally, the all junctions files for more sensitive event detection.

Output Files

junction2events produces two files:

- Annotated Junctions (`<prefix>AnnotatedJunctions.tsv`): a file containing the junctions from the primary input file, with added event annotations for each junction.
- Events (`<prefix>Events.tsv`): a file containing the events composed from junctions in the input file.

Annotated Junctions

When generating an annotated junction output, junction2events adds the columns listed in Table 17 for input files from data generated before Analysis Pipeline version 2.0.

Table 17: Additional Columns Included in Annotated Junctions Output

	Column Name	Description
1	EventId	Integer ID that links the junction file to the event file.
2	Type	Type of the event that caused the junction.
3	RelatedJunctions	Semicolon-separated list of other junctions that were grouped with this junction

Figure 26: Example AnnotatedJunctions.tsv Output

LeftPosition	LeftStrand	LeftLength	RightChr	RightPosition	RightStrand	RightLength	StrandConsistent	Interchromosomal	Distance	DiscordantMatePairAlignments	JunctionSequenceResolved	TransitionSequence	TransitionLength	LeftRepeatClassification
964098	+	594	chr1	964525	+	278	Y	N	427	19	Y		0	Self chain;Tandem period 61
1070130	-	88	chr1	1070188	-	58	Y	N	58	7	Y		0	Self chain;Tandem period 58
1232599	+	449	chr1	1233185	+	401	Y	N	586	11	Y		0	Self chain;Tandem period 226;Tandem period 46
2024555	+	617	chr1	2027385	+	561	Y	N	2830	84	Y		0	Self chain
2052874	-	162	chr1	2053146	-	204	Y	N	272	3	Y	AG	2	MLT1A:LTR:ERVL-MaLR;Self chain;Tandem period 22

Figure 26: Example AnnotatedJunctions.tsv Output (continued)

>Id	RightRepeatClassification	LeftGenes	RightGenes	XRef	DeletedTransposableElement	KnownUnderrepresentedRepeat	FrequencyInBaselineGenomeSet
3863	Self chain	NM_198576	NM_198576				0.7
1187	Self chain;Tandem period 58						0
3862	Self chain;Tandem period 226	NM_030649	NM_030649				0.25
3861	C-rich:Low_complexity:Low_complexity;Self chain;Tandem period 10;Tandem period 15;Tandem period 25;Tandem period 35	NM_001033581;NM_002744	NM_001033581;NM_002744				0
1186	(ATGGTG)n:Simple_repeat:Simple_repeat;Self chain;Tandem period 18;Tandem period 36;Tandem period 54	NM_001033581;NM_001033582;NM_002744	NM_001033581;NM_001033582;NM_002744				0

Figure 26: Example AnnotatedJunctions.tsv Output (continued)

>Id	AssembledSequence	EventId	Type	RelatedJunctions
3863	cctcatgggccaagggcacccacagccacgCCACCCTCTCCGAAGGAACC GAGCCCCAGCCCCTCGTGGGCCAAGGGCGCCCACAGCCACGCCACCCTCT CCCAAGGAACCGAGCCCCAGCCCCTCGTGGGCCAAGGGCGCCCACAGCCA CGCCACCCTTTCCGAAGGAACCGAGCCCCAGCCCCTCTGGGGCCTGCCAA TTGCCAGAGAGCCCCAGtgctccacccactccaggccccaaccccca	3284	deletion	
1187	gagaggggctgcagcctcagatggcgaggaAGCCACACCCCTCACGGTGC CCCCTCCTGAGAGGGGCTGCagcctcagatggcgaggaagccacacccct	983	tandem-duplication	
3862	cacacacaggcgtgtgcacgtgtgtggggcAGGGGCCATCCCCAGTGGCA CGTGTGTGTGTGCACAGGCGCGGGGCAGGGGCCATCCCCGGTGGCACATG TGTGCACGGGCTTGGGGCAGGGGCCATCCCCGGTGGCACGTGTGTGTGTG CACGGGCTTGGGGCAGGGgcaccaagggccacacctcgctatagcaac	3283	deletion	
3861	tttcaacaggaactcaagagaaaagttcatGCTTGTCCCCTCACCTGTTT gactctgctgtgccaggggccagggcaggg	3282	deletion	
1186	caccgtgaccctaactaactaccgccatgaCCCTAACTACCACCATGACC CTAACTACAGCCGTGACCCTAACTACCACCATGACCCTAACTACCACCAT GACCCTAACTACCACCGTGACCCTAACTACCACCATGACCCTAACTACCA CCGTGACCCTAACCACCACCGTGACCTTAACTACCACCGTCACCCTAACT ACAGCCATAGCcctaactactgatataaccctaactactgc	982	complex	1185

Events

Table 18: Column Descriptions for junctions2events Output

	Column Name	Description
1	EventId	Identifier for the event. This consists of positive integers. Event Ids are consistent across all junction files for a given assembly.
2	Type	Structural rearrangement composed of one or more junctions. Possible values include: `artifact`, `complex`, `deletion`, `tandem-duplication`, `probable-inversion`, `inversion`, `distal-duplication`, `distal-duplication-by-mobile-element`, and `interchromosomal`. See Table 19 for description of each event type.
3	RelatedJunctionIds	Junction identifier(s) of junctions that the event is composed of. Identifiers are semi-colon separated in cases where an event is represented by multiple junctions.
4	MatePairCounts	A number expressing the amount of DNB support available for each junction that the event is composed of. Numbers are semi-colon separated in cases where an event is represented by multiple junctions. They are in the order in which junction identifiers are listed in the *RelatedJunctionIds* field.
5	FrequenciesInBaselineGenomeSet	Frequency that the junction(s) is detected in set of baseline genomes. Numbers are semi-colon separated in cases where an event is represented by multiple junctions. They are in the order in which junction identifiers are listed in the *RelatedJunctionIds* field.
6	OriginRegionChr	Chromosome name in text: `chr1`, `chr2`,..., `chr22`, `chrX`, `chrY`. The mitochondrion is represented as `chrM`. The pseudoautosomal regions within the sex chromosomes X and Y are reported at their coordinates on chromosome X.

	Column Name	Description
7	OriginRegionBegin	Reference coordinate specifying the start of the region where the indicated event is likely to have originated. The coordinate uses the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
8	OriginRegionEnd	Reference coordinate specifying the end of the region where the indicated event is likely to have originated. The coordinate uses the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
9	OriginRegionLength	The distance between the left-most mate read and the right-most mate read in the junction cluster(s) representing the event at the origin site.
10	OriginRegionStrand	Strand ("+" or "-") of the indicated event at the origin site.
11	DestinationRegionChr	Chromosome name in text: `chr1`, `chr2`,..., `chr22`, `chrX`, `chrY`. The mitochondrion is represented as `chrM`. The pseudoautosomal regions within the sex chromosomes X and Y are reported at their coordinates on chromosome X. Values are only present for the following events: inversion, distal-duplication, distal-duplication-by-mobile-element, and interchromosomal.
12	DestinationRegionBegin	Reference coordinate specifying the start of the region where the indicated event is likely to have been inserted. The coordinate uses the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information. Values are only present for the following event types: `inversion`, `distal-duplication`, `distal-duplication-by-mobile-element`, and `interchromosomal`.
13	DestinationRegionEnd	Reference coordinate specifying the start of the region where the indicated event is likely to have been inserted. The coordinate uses the half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information. Values are only present for the following event types: `inversion`, `distal-duplication`, `distal-duplication-by-mobile-element`, and `interchromosomal`.
14	DestinationRegionLength	The distance between the left-most mate read and the right-most mate read in the junction cluster(s) representing the event at the destination site.
15	DestinationRegionStrand	Strand ("+" or "-") of the indicated event at the destination site.
16	DisruptedGenes	Gene(s) overlapping at least one of the junction section positions of the event.
17	ContainedGenes	Gene(s) that are completely contained in event.
18	GeneFusions	Junction that appears to either 1) connect two different genes (for example, A and B) in a strand-consistent manner or 2) connect upstream region of gene A to an intact gene B. In the former case, fusion event is described as `A/B`, where `A` and `B` are gene symbols. In the latter case, fusion event is described as `TSS-UPSTREAM[A]/B`, where A and B are gene symbols.
19	RelatedMobileElement	For duplication events caused by a mobile element, this column contains the description of the element in the format: `Family:Name:DivergencePercent` For example: `L1:L1HS:0.5`. Information for transposed locations in the reference genome is taken from the RepeatMasker track from UCSC Genome Browser track.
20	MobileElementChr	Chromosome name in text: `chr1`, `chr2`,..., `chr22`, `chrX`, `chrY`. The mitochondrion is represented as `chrM`. The pseudoautosomal regions within the sex chromosomes X and Y are reported at their coordinates on chromosome X.

	Column Name	Description
21	MobileElementBegin	Coordinate specifying the start of the consensus sequence of the specified mobile element. Uses half-open, zero-based coordinate system.
22	MobileElementEnd	Coordinate specifying the end of the consensus sequence of the specified mobile element. Uses half-open, zero-based coordinate system. See "Sequence Coordinate System" for more information.
23	MobileElementStrand	Strand ("+" or "-") of the indicated mobile element.

Table 19: Event Types

Type	Description
artifact	Event is caused by a flaw in the reference.
complex	Event involves multiple junctions and does not fit the pattern of any simple event type.
deletion	Deletion of the sequence described by the Origin columns.
tandem-duplication	Tandem duplication of the origin sequence.
probable-inversion	Inversion of the origin sequence that is confirmed from one side of the inversion only.
inversion	Inversion of the origin sequence replacing the sequence described by the Destination columns, confirmed from both sides.
distal-duplication	Copy of the origin sequence into the area described by the Destination columns.
distal-duplication-by-mobile-element	Copy of the origin sequence caused by a known active mobile element.
interchromosomal	Isolated junction between different chromosomes.

Figure 27: Example Events.tsv Output

>EventId	Type	RelatedJunctionIds	MatePairCounts	FrequenciesInBaselineGenomeSet
3	Inversion	131;4804	136;126	1.00;1.00
4	Complex	137;4805	152;152	0.40;0.40
5	complex	197;2428;4868	53;126;96	0.50;0.50;0.50
6	probable-inversion	203	50	1
7	complex	204;1242	13;26	0.90;0.00

Figure 27: Example Events.tsv Output (continued)

>EventId	OriginRegionChr	OriginRegionBegin	OriginRegionEnd	OriginRegionLength	OriginRegionStrand	DestinationRegionChr	DestinationRegionBegin	DestinationRegionLength	DestinationRegionEnd	DestinationRegionStrand	DisruptedGenes	ContainedGenes	GeneFusions
3	chr21	27374153	27374699	546	-	chr21	27374158	27374705	547	+	APP		
4													
5													
6	chr17	41381589	41463570	81981	-						LOC100130581		
7											PUS1		

Example

To identify somatic structural variants, first run junctiondiff to identify junctions present only in the tumor (generating a file called `somaticJunctionsTumor.tsv`). Then use junctions2events to interpret the somatic junctions, identifying putative somatic structural events. The file `allJunctionsBeta-GS00001-DNA_A01_1120_37-ASM.tsv` (abbreviated to `allJunctions...0_37-ASM.tsv` in the example) is the list of all junctions predicted in a given sample.

```
cgatools junctions2events \
--beta \
--reference build37.crr
--output-prefix eventOutput_ \
--junctions somaticJunctionsTumor.tsv \
--all-junctions /GS00302-DNA_A01_1120_37-ASM/GS00302-DNA_A01/ASM/SV/allJunctions...0_37-ASM.tsv \
--gene-data gene37.tsv.gz \
--repmask-data rmsk37.tsv.gz \
```

Note that with the release of Analysis Pipeline version 2.0, genomes sequenced as part of the Cancer Sequencing Service have somatic junctions reported as part of the genome package. These files can be found within the SV subfolder within the ASM directory for the tumor genomes.

Reference Tools

At times, CGA Tools uses the reference sequence in a random-access manner. The most common reference sequence format, FASTA, is not ideal for processing tasks that require random access because the entire sequence must be read into memory at the start of the program, and this memory cannot be shared among processes.

CRR File Format

CGA Tools uses its own file format, Compact Randomly Accessible Reference (CRR), to represent a reference sequence. The CRR file format stores two bits per base of reference, plus lookup tables to resolve regions of the reference that are represented by ambiguous IUPAC codes. CRR files are memory mapped, so that processes can share a reference, and the overall memory requirement due to the reference for all processes is less than 1 GB. The CRR file format does not preserve character case—FASTA reference files often use case to denote the region's repeat status—and considers all the bases described in the reference FASTA sequence as upper case.

See "Obtaining Ancillary Files for Use with CGA Tools" for downloading instructions.

FASTA Reference Sequences

Complete Genomics supports two references. The first, which we refer to as "build 36," consists of the assembled nuclear chromosomes from NCBI build 36 (without unplaced or alternate loci) plus Yoruban mitochondrion NC_001807.4. This assembly is also known as UCSC hg18. The second reference, which we refer to as "build 37," consists of the assembled nuclear chromosomes from GRCh37 (without unplaced or alternate loci), plus the Cambridge Reference Sequence for the mitochondrion (NC_012920.1). An alternative assembly using GRCh37 and another mitochondrial sequence, known as UCSC hg19, is not compatible with CGA Tools and should not be used.

See "Obtaining Ancillary Files for Use with CGA Tools" for downloading instructions.

fasta2ccr

Converts input FASTA sequences into a single reference CRR file. The following sections describe its behavior:

Synopsis

```
cgatools fasta2crr --help
       --input <fasta_file_list>
       --output <crr_file>
       --circular <chromosome_list>
```

Description

This tool converts input FASTA sequences into a single reference CRR file.

We recommend that you download pre-made reference CRR files for human genome reference builds 36 and 37. See "Obtaining Ancillary Files for Use with CGA Tools" for download instructions.

You can also download FASTA files containing the entire genome sequence for both builds if you would like to generate the reference CRR file yourself. Separate files for each chromosome are also accepted as input.

Important: Complete Genomics "build37" consists of the assembled nuclear chromosomes from GRCh37 (not unplaced or alternate loci), plus the Cambridge Reference Sequence for the mitochondrion (NC_012920.1). This assembly (though with an alternate mitochondrial sequence) is also known as UCSC hg19.

Customers who build CRR files using fasta2crr must use the correct mitochondrial sequence. CRR files generated using the UCSC hg19 FASTA files are incompatible with CGA Tools because they contain a different mitochondrial sequence.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--input <fasta_file_list>`	The input FASTA files. You can use a positional argument to specify the files or you can omit this option if using `STDIN`). Order is important: take care to specify the FASTA files in chromosome order. To work with human Complete Genomics data, the chromosome files should be in the following order, where spaces are used to separate the files: `chr1`…`chr22` `chrX` `chrY` `chrM` The entire reference human genome in one FASTA file is available on the Complete Genomics FTP site. See "Obtaining Ancillary Files for Use with CGA Tools" for download information. Note that compressed (.bz2) files are accepted as input.
`--output <crr_file>`	The output CRR file. If omitted, the output is sent to `STDOUT`.
`--circular <chromosome_list>`	A comma-separated list of circular chromosome names. If this option is omitted, `chrM` is used as default.

Input Files

The input file(s) should be in the FASTA file format. You may have separate FASTA files for each chromosome or one large file that contains all of the FASTA sequences for each chromosome. Note that fasta2crr will accept compressed files (.bz2) as input. Figure 28 shows an example input file.

Figure 28: fasta2crr Input File Example

```
>chr1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
…
TTTGCTGTTCCTGCATGTAGTTTAAACGAGATTGCCAGCACCGGGTATCATTCACCATTTTTCTTTTCGTTAACT
TGCCGTCAGCCTTTTCTTTGACCTCTTCTTTCTGTTCATGTGTATTTGCTGTCTCTTAGCCCAGACTTCCCGTGT
CCTTTCCACCGGGCCTTTGAGAGGTCACAGGGTCTTGATGCTGTGGTCTTCATCTGCAGGTGTCTGACTTCCAGC
AACTGCTGGCCTGTGCCAGGGTGCAAGCTGAGCACTGGAGTGGAGTTTTCCTGTGGAGAGGAGCCATGCCTAGAG
TGGGATGGGCCATTGTTCATCTTCTGGCCCCTGTTGTCTGCATGTAACTTAATACCACAACCAGGCATAGGGGAA
AGATTGGAGGAAAGATGAGTGAGAGCATCAACTTCTCTCACAACCTAGGCCAGTAAGTAGTGCTTGTGCTCATCT
…
>chrY
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
…
>chrM
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTTCGTCTGGGGGGTGTG
CACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTCCTGCCTCATT
…
```

Output Files

A CRR reference file, which is a custom format that is compact and enables random access of the reference.

Example

Suppose that you would like to create a CRR file for build 37 of the human genome. You can download the genome in FASTA format from the Complete Genomics FTP site. (See "Obtaining Ancillary Files for Use with CGA Tools".) The following command will generate a reference CRR file called `build37.crr`.

```
cgatools fasta2crr \
--input build37.fa.bz2 \
--output build37.crr
```

crr2fasta

Generates a FASTA file with the complete sequence of the input reference CRR file. The following sections describe its behavior:

- Synopsis
- Description
- Command Line Options
- Input Files
- Output Files
- Example

Synopsis

```
cgatools crr2fasta --help
       --input <crr_file>
       --output <fasta_file>
       --line-width <width>
```

Description

This tool converts reference CRR files into the FASTA file format.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--input <crr_file>`	The full or relative path to the reference CRR file. You can use a positional argument to specify the CRR file.
`--output <fasta_file>`	The full or relative path to the output FASTA file. If omitted, the output is sent to `STDOUT`.
`--line-width <width>`	The maximum width of a line of sequence. Defaults to 50 characters.

Input Files

A CRR reference file.

Output Files

crr2fasta generates a FASTA file with the complete sequence of the reference CRR that you provide as input. Figure 29 shows an example of the output.

Figure 29: crr2fasta Output

```
>chr1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
…
>chrM
…
TTTGCTGTTCCTGCATGTAGTTTAAACGAGATTGCCAGCACCGGGTATCACACCATCCTCCGTGAAATCAATATCCCGCACAA
GAGTGCTACTCTCCTCGCTCCGGGCCCATAACACTTGGGGGTAGCTAAAGTGAACTGTATCCGACATCTGGTTCCTACTTCAG
GGTCATAAAGCCTAAATAGCCCACACGTTCCCCTTAAATAAGACATCACGATG
```

Example

To output the entire sequence of the reference genome to a FASTA file named `build37.fa`, you can run the following command:

```
cgatools crr2fasta \
--input /ReferenceFiles/build37.crr \
--output build37.fa
```

decodecrr

Retrieves the sequence for a given range of a chromosome. The following sections describe its behavior:

Synopsis

```
cgatools decodecrr --help
       --reference <crr_file>
       --output <output_file>
       --range <range_specification>
```

Description

This command retrieves the sequence for a given range of a chromosome.

Command Line Options

Option	Description
`-h` or `--help`	Print command-line help.
`--reference <crr_file>`	The full or relative path to the reference CRR file. You can use a positional argument to specify the CRR file.
`--output <output_file>`	The output FASTA file. If omitted, output is sent to `STDOUT`.
`--range <range_specification>`	The range of bases to print, formatted as `chr,begin,end` or `chr:begin-end`.

Input Files

A CRR reference file, which is a custom format that is compact and enables random access of the reference.

Output Files

The command decodecrr extracts only the sequence from the range specified and send the information to standard out. Figure 30 shows the output format.

Figure 30: decodecrr Output Example

```
ACCCCGTCTCTACAATAAATTAAAATATTAGCTGGGCATGGTGGTGTGTGCTTGTAGTCCCAGCTACTTGGCGGGCTGAGGT
GGGAGAATCATCCAAGCCTTGGAGGCAGAGGTTGCAGTGAGCTGAGATTGTGACACTGCACTCCAGCCTGGGAGACAGAGTG
AGACTCCTACTCAAAAAAAAACAAAAAACAAAAAACAAACCACAAAACTTTCCAGGTAACTTATTAAAACATGTTTTTTGTT
TGTTTTGAGACAGAGTCTTGCTCTGTCGCCCAGGCTGGAGTGCAGTGGAGCAATCTCAGCTCACTGCAAGCTCCGCCTCCCG
GGTTCACACCATTCTCCTGCCTCAGCCTCCCGAGTAGCTAGGACTATAGGCACCCGCCACCACGCCCAGCTTATTTTTTTTG
TATTTTTTAGTAGAGACGGGGTTTCATCGTGTTAGCCAGGATGGTCTCGATCTCCTGACCTCGTGATCCGCCCACCTCAGCC
TCCCAAAG
```

Example

To extract the sequence for the PCDH7 gene on chromosome four, at coordinates 30722030 to 30726957, you would use the following command:

```
cgatools decodecrr --reference /home/complete/ref37/build37.crr \
–range chr4:30722030-30726957
```

decodecrr extracts only the sequence from the range specified. Figure 30 shows the output.

listcrr

Lists the chromosomes, contigs, or regions of ambiguous sequence within the reference. The following sections describe its behavior:

Synopsis

```
cgatools listcrr –-help
       --reference <crr_file>
       --output <output_file>
       --circular <chromosome_list>
       --mode <output_type>
       --min-contig-gap-length <length>
```

Description

This command lists the chromosomes, contigs, or regions of ambiguous sequence within the reference, depending on the parameters.

The contigs described by listcrr are defined to be the contiguous sequence bases separated by at least `min-contig-gap-length` no-call bases, where `min-contig-gap-length` defaults to 50. The default contigs correspond to the notion of contig employed in the Complete Genomics data, such as reference scores. The default mode is chromosome.

Command Line Options

Option	Description
`-h or --help`	Print command-line help.
`--reference <crr_file>`	The full or relative path to the reference CRR file. You can use a positional argument to specify the CRR file.
`--output <output_file>`	The full or relative path to the output file. If omitted, the output is sent to `STDOUT`.
`--circular <chromosome_list>`	A comma-separated list of circular chromosome names. If this option is omitted, `chrM` is used as default.
`--mode <output_type>`	The output type. Specify `arg` as one of: ▪ `chromosome`: (Default) Lists each chromosome, its length, whether or not it is circular, and the Md5 checksum value. ▪ `contig`: List the contigs and gaps in the reference sequence and their locations. ▪ `ambiguity`: Lists the regions of the reference that are ambiguous (N).
`--min-contig-gap-length <length>`	Minimum length of the gap between reference contigs, for `mode=contig`. If this option is omitted, the gap defaults to 50.

Input Files

A CRR reference file, which is a custom format that is compact and enables random access of the reference.

Output Files

After you have successfully downloaded a build 37 CRR file (or converted the downloaded reference into CRR using fasta2crr) for use with Complete Genomics data, the listcrr command in `mode=chromosome` returns the output shown in Figure 31.

Figure 31: listcrr Output for Build 37 with mode=chromosome

```
ChromosomeId Chromosome    Length Circular Md5
           0       chr1 249250621    false 1b22b98cdeb4a9304cb5d48026a85128
           1       chr2 243199373    false a0d9851da00400dec1098a9255ac712e
           2       chr3 198022430    false 641e4338fa8d52a5b781bd2a2c08d3c3
           3       chr4 191154276    false 23dccd106897542ad87d2765d28a19a1
           4       chr5 180915260    false 0740173db9ffd264d728f32784845cd7
           5       chr6 171115067    false 1d3a93a248d92a729ee764823acbbc6b
           6       chr7 159138663    false 618366e953d6aaad97dbe4777c29375e
           7       chr8 146364022    false 96f514a9929e410c6651697bded59aec
           8       chr9 141213431    false 3e273117f15e0a400f01055d9f393768
           9      chr10 135534747    false 988c28e000e84c26d552359af1ea2e1d
          10      chr11 135006516    false 98c59049a2df285c76ffb1c6db8f8b96
          11      chr12 133851895    false 51851ac0e1a115847ad36449b0015864
          12      chr13 115169878    false 283f8d7892baa81b510a015719ca7b0b
          13      chr14 107349540    false 98f3cae32b2a2e9524bc19813927542e
          14      chr15 102531392    false e5645a794a8238215b2cd77acb95a078
          15      chr16  90354753    false fc9b1a7b42b97a864f56b348b06095e6
          16      chr17  81195210    false 351f64d4f4f9ddd45b35336ad97aa6de
          17      chr18  78077248    false b15d4b2d29dde9d3e4f93d1d0f2cbc9c
          18      chr19  59128983    false 1aacd71f30db8e561810913e0b72636d
          19      chr20  63025520    false 0dec9660ec1efaaf33281c0d5ea2560f
          20      chr21  48129895    false 2979a6085bfe28e3ad6f552f361ed74d
          21      chr22  51304566    false a718acaa6135fdca8357d5bfe94211dd
          22       chrX 155270560    false 7e0e2e580297b7764e31dbc80c2540dd
          23       chrY  59373566    false 1e86411d73e6f00a10590f976be01623
          24       chrM     16569     true c68f52674c9fb33aef52dcf399755519
```

When `mode=contig` is specified, listcrr displays the contigs and gaps in the reference sequence. An excerpt of the output is shown in Figure 32:

Figure 32: listcrr Output with mode=contig

```
ChromosomeId Chromosome   Type    Offset    Length
           0       chr1    GAP         0     10000
           0       chr1 CONTIG     10000    167417
           0       chr1    GAP    177417     50000
           0       chr1 CONTIG    227417     40302
           0       chr1    GAP    267719     50000
           0       chr1 CONTIG    317719    153649
           0       chr1    GAP    471368     50000
           0       chr1 CONTIG    521368   2112852
           0       chr1    GAP   2634220     50000
           0       chr1 CONTIG   2684220   1161048
           0       chr1    GAP   3845268    150000
           0       chr1 CONTIG   3995268   9057730
```

When `mode=ambiguity` is specified, listcrr displays the regions of the reference where there is ambiguity. An excerpt of the output is shown in Figure 33:

Figure 33: listcrr Output with mode=ambiguity

```
ChromosomeId Chromosome Code    Offset    Length
           0       chr1    N         0     10000
           0       chr1    N    177417     50000
           0       chr1    N    267719     50000
           0       chr1    N    471368     50000
           0       chr1    N   2634220     50000
           0       chr1    N   3845268    150000
           0       chr1    N  13052998     50000
           0       chr1    N  13219912    100000
           0       chr1    N  13557162     50000
           0       chr1    N  17125658     50000
           0       chr1    N  29878082    150000
           0       chr1    N 103863906     50000
           0       chr1    N 120697156     50000
           0       chr1    N 120936695    150000
```

Example

Suppose you would like to display the regions of the reference sequence where there is ambiguity (called 'N'). You would enter:

```
cgatools listcrr --reference /home/complete/build37.crr --mode ambiguity
```

Appendix

snpdiff Algorithm

The algorithm employed by snpdiff is as follows, for each allele:

- Find the call in the variant file that overlaps the position in question. Use this call alone to determine the base call for the position in question.
- Walk the *alleleSeq* column of the call from the right and left until reaching the position in question. For each direction, any of the following outcomes may be reached:
  - WALK_OK – The position in question was reached.
  - WALK_EOS – The end of alleleSeq was reached before getting to the position in question.
  - WALK_INCOMPATIBLE – A base call incompatible with the reference base was found at some position before reaching the position in question.
  - WALK_LENGTH_NOCALL – A length no-call (represented by "?") is discovered before reaching the position in question.
- Combine the results of the walk from the right and left to determine the result. The results are combined by the following rules:
  - If the walk from the left and right both end up at the position of interest (WALK_OK):
    - If the base calls discovered by the two walks are in conflict, declare a larger variation (".").
    - If the base calls discovered by the two walks are consistent and at least one is called, use the base call.
    - If both walks end up with a no-call ("N"), the result is no-call.
  - If only one walk ends up at the position of interest (WALK_OK), use the base discovered by that walk.
  - If neither walk ends up at the position of interest, then:
    - If either walk ends up as WALK_LENGTH_NOCALL, mark the position as no-call ("N").
    - If either walk ends up as WALK_EOS, mark the position as deleted ("-").
    - Otherwise, mark the position as a larger variant (".").

Figure 34: Algorithm Logic from snpdiff

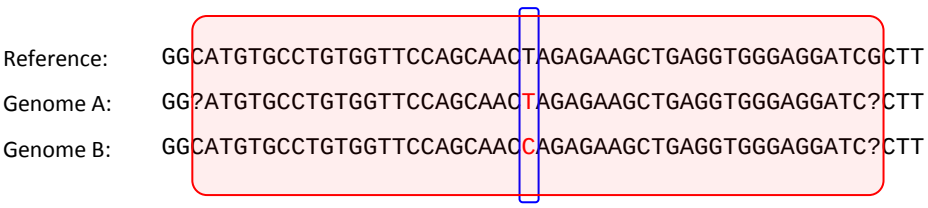
reference	alleleSeq	Walk L->R	Walk R->L	Outcome
A	C	WALK_OK: C	WALK_OK: C	C
ACGTACGT	ACGTACGT	WALK_OK: T	WALK_OK: T	T
G	CC	WALK_OK: C	WALK_OK: C	C
G	CG	WALK_OK: C	WALK_OK: G	.
ACGT	AGGN	WALK_OK: G	WALK_OK: G	G
ACGT	AGG?	WALK_OK: G	WALK_LENGTH_NOCALL	G
ACGT	?GG?	WALK_LENGTH_NOCALL	WALK_LENGTH_NOCALL	N
ACGT	CGGT	WALK_INCOMPATIBLE	WALK_OK: G	G
ACGT	CGGG	WALK_INCOMPATIBLE	WALK_INCOMPATIBLE	.
CACACAC	CAC	WALK_EOS	WALK_EOS	-

calldiff Algorithm

The calldiff tool compares two variant files to determine where and how the two genomes differ. To achieve this, it first gathers variants into superloci, which may account for several nearby variants. It compares the genomes for each superlocus then refines the comparison result to get call-level and locus-level detail.

If the superloci are too small, superlocus comparison tends to be overly sensitive to canonical alignment. However, if superloci are too large, superlocus comparison tends to allow any sequence from one genome to match in a gap of unknown sequence in the other genome. As an example of a superlocus that is too large, suppose we had the sequence from a haploid chromosome of two genomes shown in Figure 35:

Figure 35: Example of a Superlocus that is too Large



The superlocus is circled in red. Genomes A and B are consistent when considering the red superlocus as a whole because the called sequence between the "?" characters in Genome A may be aligned to the "?" character of Genome B.

When considering the red superlocus in Figure 35, and when interpreting the meaning of the calls literally, we can see that all the called bases between the "?" characters in Genome A may be aligned to the "?" character of Genome B, and the genomes are consistent. But when considering the blue box to be the superlocus, we see that the genomes are inconsistent. In different contexts, one superlocus or the other may be preferable, but generally for most comparisons, **we would want a comparison algorithm in this case to state the inconsistency between the genomes**. To achieve this, a comparison algorithm must either be very precise about how to compare superloci or very precise about how to define a superlocus:

- *Precise about how to compare superloci:* such as when using the red superlocus, determine that there is enough high complexity and uncommon sequence between the "?" characters in Genome A that the SNP in the middle must be aligned as called.
- *Precise about how to define a superlocus:* such as always use the blue superlocus in this situation.

calldiff achieves its specificity by being precise about its superlocus definition.

To determine the superloci, calldiff begins by labeling each reference region containing a variant in either variant file as a superlocus. The superloci are then extended according to the following criteria:

1. **Circular prefix/suffix matching.** For every call whose *alleleSeq* does not contain "N" or "?", do prefix matching to the right along the reference and suffix matching to the left along the reference of both the alleleSeq and the reference sequence, such that the superlocus extension does not exceed P bases (the P limit is necessary to limit the superlocus size for pathological situations). For example, if the call is for an insertion of "ACGT" and the reference sequence directly to the right is "ACGA", three prefix bases of the alleleSeq can be matched to the reference sequence directly to the right, indicating that an equivalent insertion exists at each position in that range. So the superlocus must be extended to account for any variants within three bases to the right of the variant. Additionally, in the example above, if the sequence directly to the right of the call was "ACGTACGA", then the entire insertion of four bases can be prefix matched, and continuing along the reference, the next three bases also match

the prefix of the insertion. (This is circular prefix matching.) So the superlocus must be extended to the right by seven bases.

2. **Fixed base count.** Always extend superloci to the right and left by N bases, where N is a command-line configurable parameter. Currently, this parameter defaults to 0.
3. **Fixed count of distinct 3-mers.** Always extend by M distinct reference 3-mers to the right and left, where M is a command-line configurable parameter. In regions of low reference sequence complexity, this results in longer superloci. In regions of high reference sequence complexity, this results in shorter superloci. Currently, this parameter defaults to 4.

After the superloci have been fully extended, overlapping and abutting superloci are combined into a single superlocus.

After superloci have been found, all possible phasings consistent with the *hapLink* values in the calls are used to produce hypotheses about what the genome sequence is, for each variant file. Then each permutation of each hypothesis (one permutation for haploid, two for diploid, and six for triploid) is compared to each hypothesis of the other variant file according to a literal interpretation of their sequence. In other words, any number of bases may align against length no-calls ("?"). The best comparison is produced, such that the number of discordant haplotypes is minimized. The alleles of the best comparison are then segmented to get call-level comparison results. The call-level comparison results are defined to be no worse than the result for the allele as a whole; if a segment comparison results in a worse comparison result than the allele as a whole, the allele's comparison result is used in its place. The call-level comparison results are then used to classify the comparison of each locus as a whole.

The results of calldiff are, for each allele, a comparison classification as described in Table 6.

calldiff for Scoring Somatic Variations (beta)

Somatic variation discovery is an important use case for calldiff. calldiff will identify variations that exist only in the input file A and not in file B and assign a somatic score to each of those variants to help tease apart the true somatic mutations from false somatic mutations.

calldiff uses the scores provided in Complete Genomics variation file (*varScoreVAF*, or *totalScores* for data prior to Assembly 2.0) and the calibrated scores specified using the `--calibration-root` option to determine which somatic mutations are called with higher confidence, and provides this information as a single somatic score. The somatic score is defined as:

$$SomaticScore = -10 \log_{10} L_{som}$$

where L_{som} is the estimated likelihood ratio

$$\frac{P(somatic\ call\ is\ false)}{P(somatic\ call\ is\ true)}$$

To estimate this likelihood ratio, we use the formula:

$$L_{som} = L_A \frac{n_{TP,A}}{n_{som}} + L_B \frac{n_{TN,B}}{n_{som}}$$

Here, L_A is the likelihood ratio $\frac{\text{P(variant call in A is false)}}{\text{P(variant call in A is true)}}$, L_B is the likelihood ratio $\frac{\text{P(reference call in A is false)}}{\text{P(reference call in A is true)}}$, $n_{TP,A}$ is the number of true positives in the A genome, $n_{TN,B}$ is the number of true negative base calls in the B genome, and n_{som} is the number of true somatic mutations. This formula is derived in "Computing the Somatic Likelihood Ratio".

Note that the likelihood ratios L_A and L_B are derived in a straightforward manner from the calibration data, correcting for the count of variants present in this genome and assuming that the count of false positives in A scales as the count of bases in the genome but the count of false negatives in B scales as the count of true variants in the genome. The likelihood ratio L_A is also substantially affected by the choice of variant model, as described in "Calldiff Diploid Option".

The computation of the SomaticScore makes the following assumptions:

- The rate of somatic snp is 1 per Mb.
- The rate of somatic ins is 1 per 10Mb.
- The rate of somatic del is 1 per 10Mb.
- The rate of somatic sub is 1 per 20Mb.

Note that the rate of somatic variant is an assumed value (which is different for each *somaticCategory*), and we do not attempt to determine it empirically. With additional knowledge of the true rate of somatic mutation, it should be straightforward to scale the somatic score to better reflect this reality, according to equation (2) above. For more information on whether to use the `--diploid` option and what score cutoff to use, see the section "SomaticScore Characterization".

Computing the Somatic Likelihood Ratio

Given a locus L where genome A (the "tumor" genome) has a variant call and genome B (the "normal" genome) has a reference call, we wish to determine the likelihood that the discordance is a true difference between the genomes. We define $\boldsymbol{v_A}$ to be the condition that genome A is called variant at L, with score *varScoreA*. We define $\boldsymbol{r_B}$ to be the condition that genome B is called reference at L with score *refScoreB*.

The calibrated scores provided by Complete Genomics allow us to determine the likelihood ratios $\boldsymbol{L_A}$ and $\boldsymbol{L_B}$, which are a measure of the likelihood the variant call in genome A is false ($\boldsymbol{FP_A}$) or true ($\boldsymbol{TP_A}$) given the raw score information ($\boldsymbol{v_A}$), and the likelihood the reference call in genome B is false ($\boldsymbol{FN_B}$) or true ($\boldsymbol{TN_B}$) given the raw score information ($\boldsymbol{r_B}$). The likelihood ratios are defined as follows:

$$L_A = \frac{P(FP_A|v_A)}{P(TP_A|v_A)}$$

$$L_B = \frac{P(FN_B|r_B)}{P(TN_B|r_B)}$$

Given this, we wish to determine the somatic likelihood ratio given the raw score information for the calls:

$$L_{som} = \frac{P(not\ somatic|v_A, r_B)}{P(somatic|v_A, r_B)} = \frac{P(FP_A, TN_B|v_A, r_B) + P(TP_A, FN_B|v_A, r_B)}{P(TP_A, TN_B|v_A, r_B)}$$

$$= \frac{P(FP_A|TN_B, v_A, r_B)P(TN_B|v_A, r_B)}{P(TP_A, TN_B|v_A, r_B)} + \frac{P(FN_B|TP_A, v_A, r_B, d)P(TP_A|v_A, r_B)}{P(TP_A, TN_B|v_A, r_B)}$$

$$= \frac{P(FP_A|TN_B, v_A, r_B)P(TN_B|v_A, r_B)}{P(TP_A|TN_B, v_A, r_B)P(TN_B|v_A, r_B)} + \frac{P(FN_B|TP_A, v_A, r_B, d)P(TP_A|v_A, r_B)}{P(TN_B|TP_A, v_A, r_B, d)P(TP_A|v_A, r_B)}$$

$$= \frac{P(FP_A|TN_B, v_A, r_B)}{P(TP_A|TN_B, v_A, r_B)} + \frac{P(FN_B|TP_A, v_A, r_B)}{P(TN_B|TP_A, v_A, r_B)}$$

By Bayes' theorem we have:

$$= \frac{P(TN_B, v_A, r_B|FP_A)P(FP_A)}{P(TN_B, v_A, r_B|TP_A)P(TP_A)} + \frac{P(TP_A, v_A, r_B|FN_B)P(FN_B)}{P(TP_A, v_A, r_B|TN_B)P(TN_B)}$$

By independence assumption we have:

$$= \frac{P(TN_B, r_B|FP_A)P(v_A|FP_A)P(FP_A)}{P(TN_B, r_B|TP_A)P(v_A|TP_A)P(TP_A)} + \frac{P(TP_A, v_A|FN_B)P(r_B|FN_B)P(FN_B)}{P(TP_A, v_A|TN_B)P(r_B|TN_B)P(TN_B)}$$

$$= \frac{P(r_B|TN_B, FP_A)P(TN_B|FP_A)P(v_A|FP_A)P(FP_A)}{P(r_B|TN_B, TP_A)P(TN_B|TP_A)P(v_A|TP_A)P(TP_A)} + \frac{P(v_A|TP_A, FN_B)P(TP_A|FN_B)P(r_B|FN_B)P(FN_B)}{P(v_A|TP_A, TN_B)P(TP_A|TN_B)P(r_B|TN_B)P(TN_B)}$$

We assume the call and score in B does not depend on the A genome or call in A, and vice versa. So:

$P(r_B|TN_B, FP_A) = P(r_B|TN_B, TP_A)$ and $P(v_A|TP_A, FN_B) = P(v_A|TP_A, TN_B)$.

Thus:

$$L_{som} = \frac{P(TN_B|FP_A)P(v_A|FP_A)P(FP_A)}{P(TN_B|TP_A)P(v_A|TP_A)P(TP_A)} + \frac{P(TP_A|FN_B)P(r_B|FN_B)P(FN_B)}{P(TP_A|TN_B)P(r_B|TN_B)P(TN_B)} \quad (1)$$

We can also apply Bayes' theorem to the definition of $\boldsymbol{L_A}$ and $\boldsymbol{L_B}$ from above:

$$L_A = \frac{P(FP_A|v_A)}{P(TP_A|v_A)} = \frac{P(v_A|FP_A)P(FP_A)}{P(v_A|TP_A)P(TP_A)}$$

$$L_B = \frac{P(FN_B|r_B)}{P(TN_B|r_B)} = \frac{P(r_B|FN_B)P(FN_B)}{P(r_B|TN_B)P(TN_B)}$$

Plugging this into (1) above, we have:

$$L_{som} = L_A \frac{P(TN_B|FP_A)}{P(TN_B|TP_A)} + L_B \frac{P(TP_A|FN_B)}{P(TP_A|TN_B)}$$

We estimate $P(TN_B|FP_A) = 1$. In other words, most true reference positions in A are true reference positions in B, and most true reference positions in B are called reference. Similarly, we also estimate $P(TP_A|FN_B)$=1. Additionally, we define n_{som} to be the count of true somatic loci, $n_{\text{TP,A}}$ to be the count of true variants in genome A, and $n_{TN,B}$ to be the total count of true reference positions in B. We have:

$$P(TN_B|TP_A) = \frac{n_{som}}{n_{TP,A}}$$

$$P(TP_A|TN_B) = \frac{n_{som}}{n_{TN,B}}$$

Thus:

$$L_{som} = L_A \frac{n_{TP,A}}{n_{som}} + L_B \frac{n_{TN,B}}{n_{som}} \quad (2)$$

Calldiff Diploid Option

When using the `--diploid` option to calldiff, we use the calibration of *varScoreEAF*, and additionally assume all germline heterozygous and somatic variants are present at 50% allele fraction.

Without this option, we use the calibration of *varScoreVAF* and assume a mixture model, where half the germline heterozygous and somatic variants are present at 50% allele fraction and half are present at 20% allele fraction. This effectively increases our confidence in lower scoring variants, such as would be the case for somatic variants of low allele fraction.

To see how this works, we first define mix $= m$ to be the condition that the variants in the A genome contain a mixture of 20% allele fraction and 50% allele fraction, with m being the fraction of variants present at 20% allele fraction. We define $L_{A,mix=m}$:

$$L_{A,mix=m} = \frac{P(FP_A|v_A, mix = m)}{P(TP_A|v_A, mix = m)}$$

$$= \frac{P(v_A, mix = m|FP_A)P(FP_A)}{P(v_A, mix = m|TP_A)P(TP_A)}$$

$$= \frac{P(v_A, mix = m|FP_A)}{mP(v_A, mix = 1|TP_A) + (1-m)P(v_A, mix = 0|TP_A)} \frac{P(FP_A)}{P(TP_A)}$$

We now assume that $P(v_A, mix = m|FP_A) = P(v_A, mix = 0|FP_A) = P(v_A, mix = 1|FP_A)$. That is, the score distribution of false calls does not depend on the mixture fraction m.

$$= \frac{1}{m\frac{P(v_A, mix = 1|TP_A)}{P(v_A, mix = 1|FP_A)}\frac{P(TP_A)}{P(FP_A)} + (1-m)\frac{P(v_A, mix = 0|TP_A)}{P(v_A, mix = 0|FP_A)}\frac{P(TP_A)}{P(FP_A)}}$$

$$L_{A,mix=m} = \frac{1}{\frac{m}{L_{A,mix=1}} + \frac{1-m}{L_{A,mix=0}}}$$

The derivation is easily extended for mixtures of more than two sets of variants with differing allele fraction.

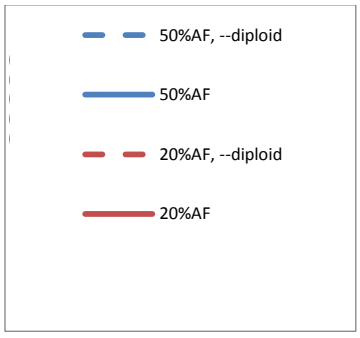
SomaticScore Characterization

Complete Genomics conducted experiments to characterize the behavior of the calldiff *somaticScore* for typical high coverage and standard coverage genomes. Under a variety of conditions, the sensitivity and specificity were recorded for each somatic score to produce ROC curves. To characterize the high coverage behavior, genomes sequenced to a depth of 110X gross coverage were used; to characterize the standard coverage behavior, genomes sequenced to a depth of 55X gross coverage were used. These numbers correspond to typical gross coverage numbers for Complete Genomics genomes for the high coverage product and standard coverage product. The Analysis Pipeline version 2.0 results are given in the plots (Figure 37 through Figure 40) and tables (Table 20 and Table 21).

For each plot, the X-axis consists of the count of somatic discordances observed in a replicate pair. The Y-axis consists of the count of dbSNP variants discovered in an *in-silico* mixture of NA19240 reads and NA12878 reads and not in a pure NA12878 sample, normalized such that a Y-value of 1 is the count of dbSNP variants discovered in pure NA19240 and not in NA12878 for the case of high coverage with no score cutoff. For the 50%AF (50% allele fraction) case, the NA19240+NA12878 mixture consisted of 100% NA19240. For the 20%AF case, the NA12940+NA12878 mixture consisted of 40% NA19240 and 60% NA12878.

As illustrated in the characterization plots below, using the `--diploid` option can slightly improve ROC for insertions, deletions, and SNPs at 50% allele fraction. However, the default behavior (without the `--diploid` option) yields substantially better ROC for SNPs at 20% allele fraction, but makes little difference for insertions, deletions, and substitutions. Additionally, the benefits of sequencing at higher coverage depth are apparent as the ROC is substantially better for high coverage than standard coverage, especially for insertions, deletions, and SNPs at 20% allele fraction.

Figure 36: Legend for All Characterization Plots



50%AF represents scenarios where somatic variant is expected to be present at 50% allele fraction.

20%AF represents scenarios where somatic variant is expected to be present at 20% allele fraction.

--diploid represents scenarios where the calibration of *varScoreEAF* is used to compute the *somaticScore* and all germline heterozygous and somatic variants are assumed to be present at 50% allele fraction.

Figure 37: High and Standard Coverage *in-silico* Mixture ROC Curve by *somaticScore* for SNPs

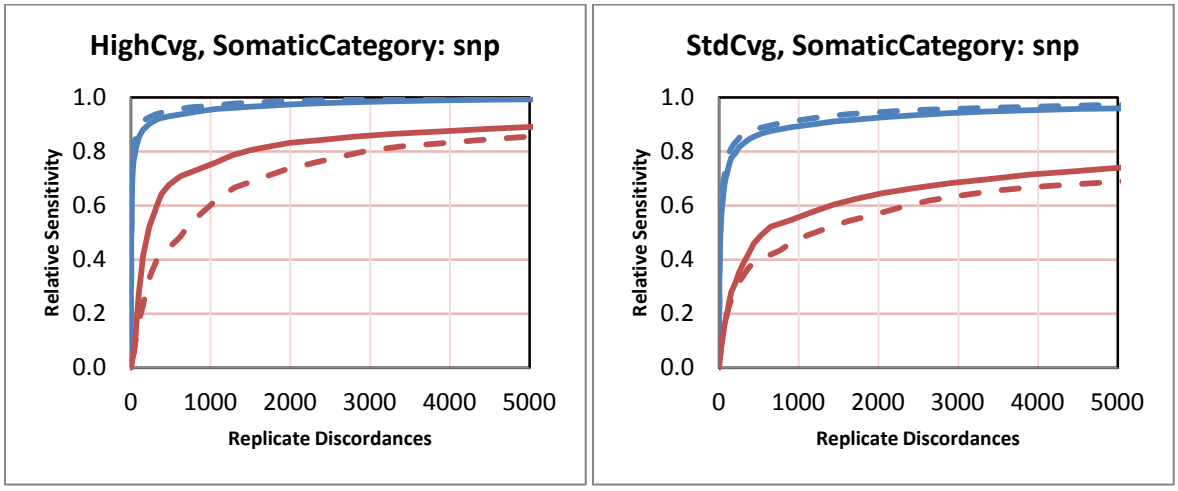


Figure 38: High and Standard Coverage *in-silico* Mixture ROC Curve by *somaticScore* for Insertions

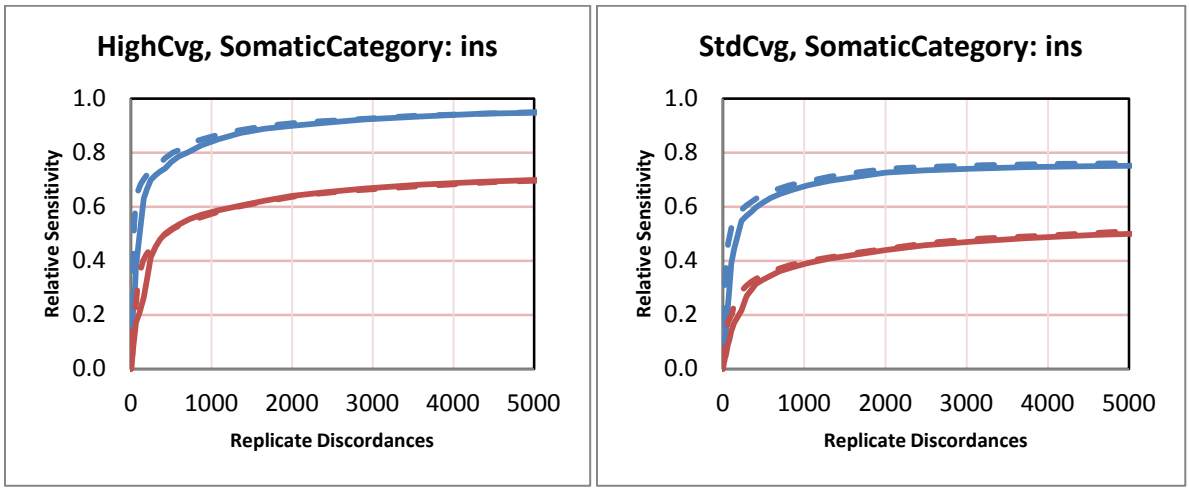


Figure 39: High and Standard Coverage *in-silico* Mixture ROC Curve by *somaticScore* for Deletions

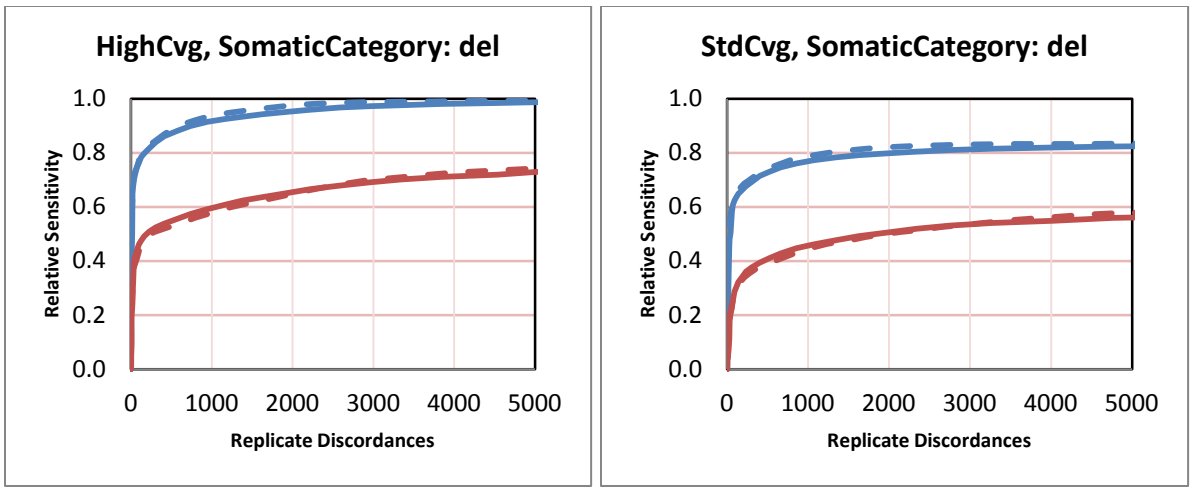
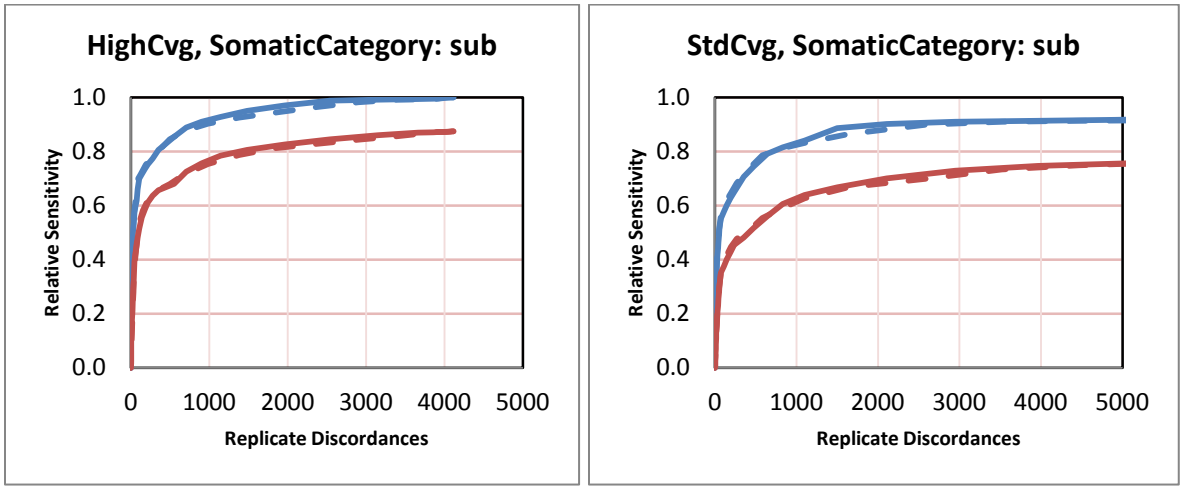


Figure 40: High and Standard Coverage *in-silico* Mixture ROC Curve by *somaticScore* for Substitutions



The data in Table 20 and Table 21 illustrate a key point. To achieve high sensitivity for somatic indel detection, a lower *somaticScore* threshold is required than for SNPs. Note that this is due in part because there is an expectation that fewer true somatic indels will be present. For example, if we expect an equal number of false somatic insertions as somatic SNPs but ten times as many true somatic SNPs as somatic insertions, we may expect to require a somatic score threshold on insertions that is 10 dB lower to achieve the same level of sensitivity. At that threshold, you may expect a higher rate of error for insertions than SNPs.

Table 20: Sensitivity and Number of Discordant Calls for High Coverage *In-Silico* Mixture

High Coverage										
			snp		ins		del		sub	
AF	Options	Somatic Score	sens	discord	sens	discord	sens	discord	sens	discord
50%AF	default	-30	0.999	10758	0.953	4427	0.98	3750	0.992	2529
50%AF	default	-25	0.997	7636	0.925	2141	0.962	2215	0.964	1134
50%AF	default	-20	0.993	4782	0.867	976	0.925	1059	0.85	410
50%AF	default	-15	0.978	2281	0.761	402	0.858	345	0.712	106
50%AF	default	-10	0.956	1017	0.659	163	0.765	79	0.567	39
50%AF	default	-5	0.919	315	0.255	29	0.679	28	0.432	7
50%AF	default	0	0.861	109	0.062	5	0.327	6	0.179	1
50%AF	--diploid	-30	0.998	5650	0.961	4370	0.988	2740	0.995	3375
50%AF	--diploid	-25	0.996	3740	0.932	1668	0.976	1760	0.975	1023
50%AF	--diploid	-20	0.991	2195	0.874	687	0.94	935	0.867	365
50%AF	--diploid	-15	0.977	1060	0.774	249	0.866	293	0.721	81
50%AF	--diploid	-10	0.953	428	0.69	86	0.765	68	0.594	32
50%AF	--diploid	-5	0.915	122	0.377	18	0.683	25	0.456	11
50%AF	--diploid	0	0.868	54	0.152	6	0.548	5	0.234	0
20%AF	default	-30	0.923	10758	0.699	4427	0.711	3750	0.861	2529
20%AF	default	-25	0.912	7636	0.65	2141	0.667	2215	0.791	1134
20%AF	default	-20	0.887	4782	0.582	976	0.602	1059	0.654	410
20%AF	default	-15	0.838	2281	0.492	402	0.522	345	0.523	106
20%AF	default	-10	0.749	1017	0.298	163	0.448	79	0.353	39
20%AF	default	-5	0.595	315	0.081	29	0.352	28	0.141	7
20%AF	default	0	0.31	109	0.003	5	0.131	6	0.002	1
20%AF	--diploid	-30	0.859	5650	0.693	4370	0.686	2740	0.869	3375
20%AF	--diploid	-25	0.808	3740	0.617	1668	0.625	1760	0.797	1023
20%AF	--diploid	-20	0.68	2195	0.534	687	0.553	935	0.641	365
20%AF	--diploid	-15	0.454	1060	0.431	249	0.466	293	0.467	81
20%AF	--diploid	-10	0.274	428	0.319	86	0.383	68	0.251	32
20%AF	--diploid	-5	0.148	122	0.103	18	0.293	25	0.071	11
20%AF	--diploid	0	0.067	54	0.021	6	0.138	5	0.004	0

Table 21: Sensitivity and Number of Discordant Calls for Standard Coverage *In-Silico* Mixture

Standard Coverage										
			snp		ins		del		sub	
AF	Options	Somatic Score	sens	discord	sens	discord	sens	discord	sens	discord
50%AF	default	-30	0.979	13933	0.757	5095	0.822	4187	0.918	2649
50%AF	default	-25	0.975	9165	0.741	2247	0.804	2090	0.896	1051
50%AF	default	-20	0.964	5298	0.703	1036	0.77	900	0.74	339
50%AF	default	-15	0.94	2597	0.626	445	0.706	311	0.573	79
50%AF	default	-10	0.899	1063	0.537	181	0.628	90	0.458	35
50%AF	default	-5	0.842	324	0.176	42	0.489	29	0.328	12
50%AF	default	0	0.745	110	0.072	7	0.112	3	0.16	3
50%AF	--diploid	-30	0.979	8166	0.765	4475	0.831	2765	0.924	3817
50%AF	--diploid	-25	0.975	4842	0.75	1792	0.821	1795	0.904	971
50%AF	--diploid	-20	0.963	2666	0.717	798	0.792	929	0.778	314
50%AF	--diploid	-15	0.939	1300	0.64	298	0.725	323	0.592	67
50%AF	--diploid	-10	0.898	472	0.567	100	0.64	87	0.483	29
50%AF	--diploid	-5	0.845	163	0.296	10	0.555	30	0.358	8
50%AF	--diploid	0	0.761	63	0.13	3	0.391	4	0.213	3
20%AF	default	-30	0.809	13933	0.504	5095	0.558	4187	0.734	2649
20%AF	default	-25	0.786	9165	0.458	2247	0.515	2090	0.648	1051
20%AF	default	-20	0.742	5298	0.395	1036	0.449	900	0.478	339
20%AF	default	-15	0.671	2597	0.322	445	0.364	311	0.345	79
20%AF	default	-10	0.558	1063	0.197	181	0.281	90	0.224	35
20%AF	default	-5	0.402	324	0.064	42	0.174	29	0.093	12
20%AF	default	0	0.215	110	0.016	7			0.038	6
20%AF	--diploid	-30	0.727	8166	0.501	4475	0.527	2765	0.75	3817
20%AF	--diploid	-25	0.666	4842	0.442	1792	0.485	1795	0.653	971
20%AF	--diploid	-20	0.564	2666	0.377	798	0.414	929	0.47	314
20%AF	--diploid	-15	0.435	1300	0.303	298	0.328	323	0.322	67
20%AF	--diploid	-10	0.33	472	0.23	100	0.241	87	0.188	29
20%AF	--diploid	-5	0.226	163	0.081	10	0.154	30	0.081	8
20%AF	--diploid	0	0.136	63	0.025	3	0.047	4	0.016	3

listvariants Algorithm

The superlocus approach to genome comparison achieves a good combination of sensitivity to genomic variation and insensitivity to canonical alignment for a small number of genomes. But as the number and variety of genomes grows, superloci can grow arbitrarily large. As shown in "calldiff," when superloci grow too large, the literal interpretation of sequence compatibility employed by calldiff tends to be insensitive to real genomic differences.

Both listvariants and testvariants can be used for many-genome comparison. The listvariants command lists all the small variants found in an arbitrary number of genomes, and the testvariants command tests each variant to determine its presence in each of the input genomes.

The listvariants command merges and lists all the fully called mutations from a set of variant files (that is, each line from each variant file that is fully called and inconsistent with the reference). listvariants is as specific as it can be about mutations without splitting up called mutations from the variant file. For example:

Reference:	CGAATTACAT
Allele 1:	CGCATTATAT
Allele 2:	CGAATTACAT

In this case, suppose the variant file listed this sequence as two SNP mutations with the same *hapLink* to indicate they are on the same haplotype. In this case, listvariants also lists the two SNPs separately. In this way, the many-genome comparison can be very specific about where genomes differ, although it loses information about which variants occurred on the same haplotype as other variants.

listvariants also canonicalizes any input variants it encounters before writing them to the output. It uses the leftmost variant that is equivalent to the input variant. For example:

Reference:	CG-AAAAA-CAT
Alternative 1:	CGAAAAAA-CAT
Alternative 2:	CG-AAAAAACAT

The two alternatives above have the same sequence, but have different alignments against the genome. If the input genomes list both insertions, the alternative 2 alignment is canonicalized (transformed) into the alternative 1 alignment because it is the leftmost alignment that describes an equivalent sequence. The two variants are then merged as equivalent, and a single output record is produced.

testvariants Algorithm

The testvariants command processes the variants listed by listvariants and writes each input record to the output, with a flag for each allele of each genome to indicate if the variant is present on that allele as listed in Table 12.

The testvariants command tests each variant against each genome independently. To do so, it first constructs a one-genome superlocus to keep track of which loci may be used in the comparison. The superlocus is constructed using the same superlocus rules as calldiff, except the "Fixed count of distinct 3-mers" used is 6 instead of 4.

After a superlocus has been found, all possible phasings consistent with the *hapLink* values in the calls are used to produce hypotheses about what the genome sequence is. testvariants chooses the phasing that results in the most 1's, then the most 0's.

To test a particular phasing, for each allele, testvariants first finds a base set of calls consisting of the minimal set of calls overlapping the variant, extended to the left and right according to the prefix/suffix matching rule of calldiff. Then testvariants compares the call sequence to the variant sequence (extended to the left and right by reference sequence), for every sequential sequence of calls in the superlocus that covers the base set of calls. This results in "1" if any compared sequence matches the variant sequence, or

"N" if any compared sequence is compatible with the variant sequence but contains no-calls. Otherwise, the result is "0".

Although the testvariants algorithm may achieve a reasonable middle ground between sensitivity to real genomic variation and insensitivity to canonical alignment for the many-genome comparison problem, the following limitations apply:

- testvariants is more sensitive to the canonical alignment than calldiff. As such, it is not the ideal tool for comparing a small number of genomes (such as 2 or 3).
- For simplicity of output file format and interpretation, testvariants does not transfer the score of the variant calls or the reference scores in the input genomes to the output file. Not having scores further limits the testvariants output for use in analyzing the genomic differences of a small number of genomes.

junctiondiff Algorithm

The junctiondiff tool finds junctions present in one genome (genome A) but not another (genome B). It addresses the following pitfalls when comparing the set of junctions present in two genomes:

- Junction coordinates are not always exact.

  When a junction's sequence cannot be resolved (indicated when the value in the *JunctionSequenceResolved* column of the junction file is "N"), the coordinates of the junction are estimations based on the expected distribution of mate gaps of the DNB reads that contribute to the discordant mate pair analysis. To resolve this issue, junctiondiff considers all junction coordinates within N bases to be equivalent. Here, N is controlled by the command-line option `--distance`.

- Junction coordinates for the same junction are not always the same.

  Just as for short indels, the same junction sequence can sometimes be described at slightly different coordinates. Additionally, small variants near the junction may cause the junction coordinates to shift slightly. This pitfall is also resolved by the `--distance` command-line option.

- Complete Genomics junction detection algorithm has low sensitivity to very short deletions.

  Slight differences in mate gap distribution or slight changes in coverage bias characteristics for the two genomes may mean that a junction that is present with good support in genome A is not called in genome B due to lack of support. To resolve this issue, junctiondiff provides the option `--minlength` to filter out junctions consistent with short deletions.

- There is low sensitivity for junctions with few discordant mate pair alignments.

  Complete Genomics junction caller requires three discordant mate pair alignments of support to call a junction. If a junction has a low expected count of discordant mate pair alignments (such as 3), whether the junction achieves sufficient support is a matter of chance. To address this issue, junctiondiff provides the option `--scoreThresholdA` to filter out junctions with few discordant mate pair alignments.

- Genome sample B may be contaminated by genome sample A.

  This often occurs, for example, if genome sample B is a tumor and genome sample A is the matched normal. For this reason, the junctiondiff tool allows you to specify a cutoff of support using the `--scoreThresholdB` option for genome B.

Representation of the Complete Genomics Data in SAM Output Format

The detailed descriptions of the SAM output in Table 22 and Table 23 and assume some familiarity with Complete Genomics data and terminology. We recommend you consult the Complete Genomics Data File Formats document, Complete Genomics , and the Drmanac et al. paper if you are mostly familiar with other Next-Gen platforms.

The description of the SAM format shown here is based on the SAM Format Specification.

Table 22: SAM Header Fields

Section	Tag	Value	Description	Example
@HD	VN	0.1.4	Version of SAM spec	
	SO	"DnbId sorted"	Sort Order. **Note:** A DNB is a clone.	
@SQ	SN	*ChromosomeName*	Sequence Name. Included for all chromosomes in the reference genome.	`SN:chr1`
	LN	*ChromosomeLength*	Length of the chromosome. Included for all chromosomes in the reference genome.	`LN:247249719`
	UR	*ReferenceFilePath*	Path to the input reference file	`UR: reference.crr`
	AS	*ASM ID*	Reference sequence ID	`AS: GRCh37`
@RG	ID	*LaneID*	Slide and Lane ID	`ID:GS08081-FS3-L02`
	SM	*SampleID*	Sample ID	`SM:GS00028-DNA-C01`
	LB	*LibraryID*	Library ID of the library	`LB:GS00433-CLS`
	PU	*LaneID*	Slide and lane ID	`PU:GS08081-FS3-L02`
	CN	"Complete Genomics"	Name of sequencing center producing the read	
	DT	*ExportDate*	Complete Genomics data analysis timestamp stored in the ***reads*** file	`DT:2010-01-21`
	PL	"Complete Genomics"	Platform/technology used to produce the read	
@PG	ID	"cgatools"	Program name	
	VN	*Version*	Program version	`VN:0.5.0`
	CL	*Command line*	Command line	Complete string

Table 23: Mapping Record Fields

Field	Value	SAM Definition	evidence2sam Usage
QNAME	*SlideID-LaneID-Chunk*: *DnbOffset*	Query Name	The QNAME value is constructed from the full lane Id (*SlideId+LaneId*), chunk number and 0-based DNB offset from the beginning of the Reads file provided as input. For example: GS08081-FS3-L02-3:244
FLAG	0x0001	The read is paired in sequencing.	The flag is always set for Complete Genomics data. The current Complete Genomics technology always produces paired reads.
	0x0002	Each fragment properly aligned according to the aligner	The flag is set to 1 if both mates are mapped.
	0x0004	The query sequence itself is unmapped.	The flag is set when there are absolutely no mappings found for this HalfDNB.
	0x0008	The mate is unmapped.	The flag is set only when there are no mappings found for this HalfDNB's mate.
	0x0010	Strand of the query	0 for forward; 1 for reverse strand.
	0x0020	Strand of the mate	0 for forward; 1 for reverse strand.
	0x0040	The read is the first read in a pair.	The flag is set if the current HalfDNB is from the 5' end of the original cloned insert.
	0x0080	The read is the second read in a pair.	The flag is set if the current HalfDNB is from the 3' end of the original cloned insert.
	0x0100	The alignment is not primary.	The flag is set if there is a better mapping of the same HalfDNB having higher value of *MAPQ* (see MAPQ in this table).
	0x0200	The read fails platform/vendor quality checks.	Always set to 0.
	0x0400	The read is either a PCR duplicate or an optical duplicate.	Always set to 0.
RNAME	*ChromosomeID* or "*"	Reference sequence NAME	Can be "*" if this HalfDNB doesn't have mappings. If the HalfDNB is not mapped itself but has a mapped mate the RNAME of the mate is reported.
POS	*Current Mapping Position* or 0	1-based leftmost POSition/coordinate of the clipped sequence	The position reported in a Mappings file record from a Complete Genomics export package offset by 1 (Complete Genomics export format reports mapping positions as 0-based). 0 is reported if there are no mappings found for this HalfDNB and there are not mapped mates. If the HalfDNB is not mapped itself but has a mapped mate the POS of the mate is reported.
MAPQ	*CG_Mapping weight*	MAPping Quality (Phred-scaled probability that the mapping position of this read is incorrect.)	The probabilities are reported in different ranges for consistent pair reads (Flag 0x0002, case 1) and for non-paired mappings. Complete Genomics does not recommended that values of consistent and inconsistent mappings be directly compared.
CIGAR	C*igarString* and GS/GQ/GC flags	Extended CIGAR string	M, N, I, D and P operations are used in the *CIGAR* field per the SAM Format Specification. Negative gaps are represented using GS/GQ/GC flags. The CIGAR sequence will ignore the negative gaps.

Field	Value	SAM Definition	evidence2sam Usage
MRNM	"=" or *ChromosomeID* or "*"	Mate Reference sequence NaMe; "=" if the same as *RNAME*	Reports "*" if there is no consistent mate found.
MPOS	*MatePosition* or 0	1-based leftmost mate POSition of the clipped sequence	Reports 0 if there is no consistent mate found.
ISIZE	*DistanceToMate* or 0	Inferred Insert SIZE	The distance between the consistent mate start position and the start position of the current HalfDNB mapping. The value is 0 if the mates are mapped to different chromosomes.
SEQ	*Sequence*	Query SEQuence; "=" for a match to the reference; n/N/. for ambiguity	The regions of overlapping bases in the negative gaps contain the bases with higher scores.
QUAL	*QualityScores*	Query QUALity; ASCII-33 gives the Phred base quality	The values are copied from the corresponding record of the Complete Genomics ***reads*** file.
TAG	GS/GQ/GC RG R2/Q2 XS	Tags	▪ GS/GQ/GC flags are used to represent Complete Genomics-specific negative wobble gaps in HalfDNBs. ▪ RG: a standard tag containing the read group name. ▪ R2, Q2: optional, standard tags. ▪ XS:I:1 the optional user-defined tag marks an SV candidate. See SAM Format Specification in "References" for the description of the flags.

Rules to Set the "not primary" Flag (0x0100)

The flag "not primary" is set for a HalfDNB mapping in the following cases:

- The mapping of a HalfDNB doesn't have a consistent mate pair mapping, and there are consistently mapped pairs found for the DNB.
- There is another mapping of the same HalfDNB having a higher *MAPQ* value.

Combining Mapping Records in SAM

1. The best mapping pair (the best score) of a DNB is reported with the "non-primary" flag set to 0. Both mappings should refer to each other as the best mates.
2. All the other mappings of that DNB have the "non-primary" flag set to 1.
3. If both HalfDNBs are mapped uniquely but not consistently, they might be reported as primary ("non-primary" flag is set to 0) if "mate-sv-candidates" option is used.
4. If only one HalfDNB is mapped, the best mapping of that HalfDNB is reported followed by a "non-mapped" mapping record of the other HalfDNB. The alignment position of the other HalfDNB is set to the same values as the mapped HalfDNB and the "non-primary" flag of the records is set to 0. The not mapped record will be marked as "not mapped" by appropriate flags. If the options "add-unmapped-mate-info" or "add-mate-sequence" are used the SAM record for the non-mapped mate is not generated.
5. All the other mappings of the mapped read from number 4 above are reported one record per mapping having "non-primary" flag set to 1.

All the not mapped reads are reported in pairs aligned to the 0 position. The alignment position is important to keep the records together while sorting and merging BAM files. If the options "add-unmapped-mate-info" or "add-mate-sequence" are used only a record for the first unmapped mate is generated to reduce the output size.

Limitations of SAM Format Conversion

- SAM does not have strong support for overlapping sub-reads that are present in Complete Genomics data. An example of an overlapping sub-read is the negatively sized intra-read gaps. To represent overlapping reads, the strongest base call is put in the SAM mapping record, and the alternative base calls are represented using the GS/GQ/GC tags of the mapping record.
- The NM tag (edit distance to reference sequence) is not currently produced by evidence2sam.
- The R2 and Q2 tags (mate sequence and quality scores) can be generated optionally.

Rules for Converting Evidence DNBs with Multiple Mappings to SAM Format

There are two situations where a DNB may have multiple mapping records present in the evidence DNB mappings provided by Complete Genomics. First, if the best hypothesis is heterozygous and contains two non-reference alleles, support is also given for the reference allele. In this case, if a DNB supports two of the three alleles equally well (or similarly well) and not the third allele, then the evidence DNB mappings contain a record showing alignment of the DNB to each of the two alleles it supports. Second, if there are two regions of the genome with similar sequence such that a DNB aligns equally well to either sequence, the DNB may be used as evidence for alleles in both regions. A post-processing step of the Complete Genomics assembly process finds such regions and no-calls them.

Because most tools that visualize SAM do not have rich features to specify an allele a DNB maps against, visualization of the duplicate mapping records present in the evidence can be confusing. For this reason, the evidence2sam tool removes duplicate DNB mappings that are nearby on the reference by default. When the option `–keep-duplicates` is used, evidence2sam reports all mappings as non-primary mappings.

The evidence2sam tool de-duplicates using the following algorithm, for each variation interval:

1. Update the read-ahead buffer to ensure it contains all evidence mapping records up to 1 kb to the right of the position of the rightmost evidence mapping record for this interval.
2. Moving from position 0 to the end of the chromosome, processing each mapping record of the current variation interval as follows:
    a. Collect all the mappings of the same DNB that belong to the current interval or mappings from different intervals that overlap the corresponding arm/both arms of the selected DNB.
    b. Run one-DNB de-duplication. This operation deletes all the collected DNB mappings from the buffer except the "best" one.
    c. Write the "best" mapping into the SAM output stream.
    d. Remove the "best" mapping from the buffer and proceed to the next mapping in the current interval.
    e. If the last mapping in the current interval has been processed, remove the mappings processed for the current interval from the read-ahead buffer.

During de-duplication, the following rules are used to determine the best mapping for a DNB:

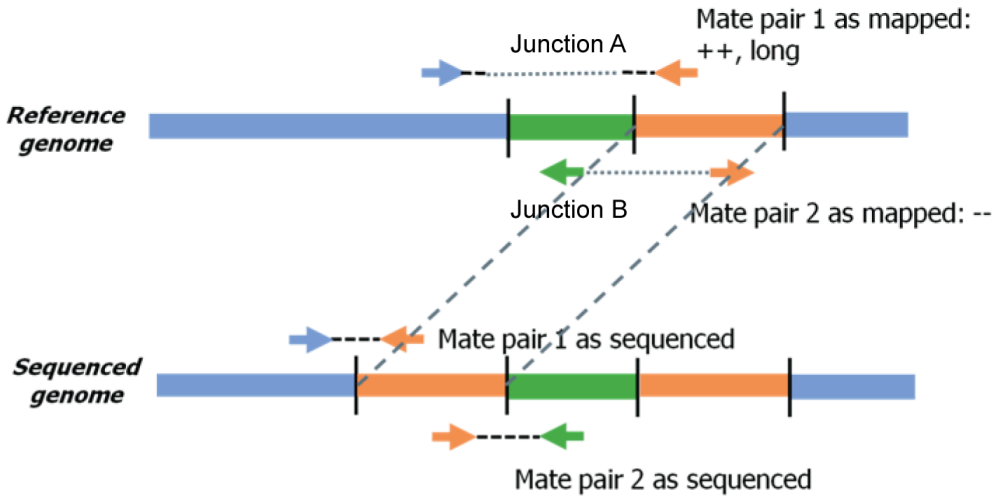
1. If several mapping records belong to the same variation interval, leave only the record that has maximum mapping quality.
2. If several mapping records belong to adjacent variation intervals (same side and strand), leave only the record that has maximum mapping quality.

3. If there are only two mapping records in the set and their different arms support different intervals, construct a composite mapping inheriting MAPQ, position in the reference, and reference alignment from a corresponding mapping record.
4. If there is still more than one mapping record in the input set, select the mapping with highest MAPQ and remove the other mappings.

junctions2events Algorithm

To understand the utility of junctions2events, consider a distal duplication event within the same chromosome as shown in Figure 41. This distal duplication event generates two junctions detected with mate pair 1 and mate pair 2.

Figure 41: Distal Duplication Event



In this example, the genomic region represented by the second orange segment of the sequenced genome is duplicated and inserted into a region upstream of the original copy. Mate pair 1 from the sequenced genome maps to the reference genome in the expected orientation, but at a distance that is greater than the expected mate gap size. Thus, Junction A represented by mate pair 1 would have both *LeftStrand* and *RightStrand* as "+" in the junction file. Mate pair 2 from the sequenced genome maps to the reference genome in an anomalous orientation: the right end maps to an earlier position in the reference genome than the left end. Thus, Junction B represented by mate pair 2 would have both strand columns as "-".

When interpreted in isolation, Junction B has the signature of a tandem duplication of the green and orange sequence. Junction A indicates a deletion of the green sequence. Both junctions seem to indicate more disruptive events than the duplication that caused them, particularly if the green sequence is long and contains many genes and the orange sequence is very short. Junction-by-junction interpretation becomes even more misleading if the duplicated orange sequence is inserted into a different chromosome such that each junction may seem to indicate a rearrangement of chromosomal arms.

Missing the detection of a true junction and detecting false junctions can lead to misinterpretation of the underlying event. To limit the chances of misinterpreting the events, junctions2events requires access to the most complete list of junctions available. However, you can specify the shorter list of junctions of interest, so an event will be listed only if it contains at least such junction.

One can deduce structural variation event types from junction data by generating an undirected graph of related junctions and then stepping through the following heuristic process:

- Junctions that are close on at least one side are considered connected. The distance threshold is controlled by the junctions2events `max-related-junction-distance` parameter.

- Connected components with more than two junctions are considered "complex" events.
- For every other junction, junction2events attempts to find a compatible junction in such a way that together they may be interpreted as a distal duplication of contiguous sequence. For example, for junction B in Figure 41, junction2events starts the search from the right position of the junction upstream, until it encounters the right position of the junction A. In general, it scans in the direction away from the break indicated by the junction side. The maximum scan distance is controlled by the `max-pairing-distance` parameter.
  Junctions are considered compatible when their sides can be paired to bound a contiguous piece of sequence from the inside, similar to the orange sequence in Figure 41, while their remaining sides bound a small piece of sequence from the outside.
- Pairs of compatible junctions are considered "inversion" events when the junctions change strand, and the sequence chunk bounded from the inside overlaps to a large degree with the sequence chunk bounded from the outside (one can think of this as the sequence being copied, inverted, and pasted over its old location). For the cases with no significant overlap, the event is classified as "distal-duplication".
- Junction pairs that are connected, but not compatible in the sense described above, are considered "complex" events.
- For isolated junctions, junction2events attempts to find a nearby mobile element that may have caused the junction by copying the adjacent sequence. The search distance is controlled by the `max-distance-to-m-e` parameter, and the list of mobile elements that are known to copy an adjacent sequence may be specified using the `mobile-element-names` parameter.
- Remaining isolated junctions that connect sequence on different chromosomes are not classified any further and are listed as "interchromosomal" events.
- Finally, the isolated junctions that have both sides on the same chromosome are interpreted based on the strands of the junction sides: Junctions with +/+ sides are classified as deletions, -/- as tandem duplications, and strand-inconsistent junctions as probable inversions. In all cases the subject sequence of the event lies between the junction side positions.

In addition to classifying the events by the type, junctions2events uses the list of known genes to annotate the events. Every event (including the "complex" events) is annotated with the list of all potentially disrupted genes; these are the genes that overlap at least one of the junction side positions for any of the junctions that were grouped into the event.

junctions2events generates the list of possible gene fusions for an event as follows:

- When a junction appears to connect two different genes (for example, A and B), it is considered a possible gene fusion (described in the file as "A/B").
- When a junction connects the region upstream of gene C to an intact gene D in a strand-consistent manner, it is annotated using "TSS-UPSTREAM[C]/D" notation; the size of the upstream region that triggers this annotation is set using the regulatory-region-length parameter.
- For the events that may indicate a copy number change of a stretch of sequence (that is, "deletion", "tandem-duplication", and "distal-duplication" events), all the genes that are completely contained in the affected sequence are included.

junctions2events produces two files:

- The list of the original junctions of interest annotated with the event type, the list of related junctions, and the unique ID of the event.
- List of the events and the corresponding gene-related annotations.

junctions2events requires two external data files.

- List of repeat annotations, described in "generatemasterVar (beta)".
- List of the known gene locations in the genome, derived from the NCBI RefSeq alignment data and reformatted to better fit the CGA Tools conventions.

You can download these files for each of the reference builds supported by Complete Genomics from ftp://ftp.completegenomics.com/AnnotationFiles/.

Sequence Coordinate System

Sequence positions in the mapping and variations files are represented in half-open, zero-based coordinates, which denote locations between successive reference base positions. A substitution or deletion of the second base (T) in the sequence of length 8 below would have a start position of 1 and an end position of 2. An insertion following the same second base would have both a start and end position of 2.

```
0       1       2       3       4       5       6       7       8
|   A   |   T   |   A   |   G   |   G   |   C   |   T   |   A   |
```

For example, the gene LIPI is located at chr21:15481137-15579254 using these coordinates.

mkvcf Translation Details

In most cases, fields in the VCF correspond exactly 1:1 to values taken from the input files. However, in some cases there are new fields derived from one or more values in the input files and there are also various reformattings. The definitions of individual fields are covered in the section on field names, above. More general observations are provided here.

masterVar Conversion

The converter aims to convert the ***masterVar*** file content to VCF by doing straightforward, 1:1 mapping as much as possible, given that the VCF is a multi-genome format where sometimes multiple loci must be merged to construct a single VCF locus. Here are some highlights of the approach to translation used in mkvcf:

- Generally, the ***masterVar*** separates multiple records within a column using a semicolon ";" and multiple parts of a record using a colon ":". These characters are generally used as separators for other reasons in VCF; the comma "," is commonly used to separate per-allele information in VCF. So where a ";" occurs in the ***masterVar***, you will see an ampersand "&" in the VCF; where a ":" occurs in the masterVar you will see a vertical bar "|" in the VCF.
- Any annotations that result in a period "." meaning no information, or a list of periods, one per allele, may be omitted from the output for compactness.
- The annotations generally only include the ***masterVar*** annotations for calls that are completely subsumed by the VCF locus. If a ***masterVar*** locus only partially overlaps the VCF record (only possible for ref-called loci and no-called loci), the annotations are not transferred to the VCF.
- The per-allele, per-genome annotations of the FORMAT fields are always lists with two values. In haploid regions, the second value is always period ".". This is required by the HQ field according to VCF 4.1 specifications. This general convention has been followed for other fields for consistency.
- Some of the annotation columns of the ***masterVar*** have to do with CNV information, and because that information is already represented by distinct records in the VCF, the information is not repeated for each small variant record.
- Non-standard sub-fields are given names with a "CGA_" prefix, so as not to collide with future standard sub-field names or other non-standard sub-field names.

Locus Definition (How We Split the Reference Genome into Records)

The genome is split into records as follows:

- Gather a superlocus, using 0 bases of extension and 0 3-mers of extension, and additionally skipping the prefix matching and suffix matching capabilities of the algorithm that defines superloci. Also

collect superloci that contain no-calls but no ref-inconsistent calls. More information about superloci can be found in "calldiff Algorithm".

- Split the superlocus up by ref-inconsistent calls of all included genomes. Overlapping ref-inconsistent calls are merged into a single VCF record. The range of any insertion, deletion, or length-changing block substitution call is extended by one base, as required by VCF. We always extend the locus one base to the left, except at the beginning of the chromosome where we extend any insertion or deletion one base to the right.
- Within the superlocus, gaps between the ranges defined above are no-calls. If the `--include-no-calls` option is on, add VCF records for these ranges as well.

Handling of Long No-Calls

If the `--include-no-calls` option is on, loci containing only no-calls are present in the output. Some very long regions of the genome are no-called, and to fit this information into VCF would normally require outputting very long strings in the REF field. As a workaround for such loci, we only output the first base of the locus in the <REF> field, we add the END tag in the <INFO> field, we output <CGA_NOCALL> for the <ALT> field, and the GT sub-field for each no-called allele is period ".".

Phasing of Merged Loci

Based on the Locus Definition, some loci from the input ***var*** file may need to be merged into a single output record. When this happens, it is possible that two loci that are not explicitly phased by hapLink information in the ***var*** file must be merged, in which case we must choose a phasing. To do this, the code first orders the genomes in ascending order by the number of possible phasings for the genome that are consistent with the hapLinks. A phasing is chosen for each genome in that order. In this way, genomes whose phasing is well-defined and have no ambiguity are picked first. When an ambiguity is discovered, the phasing for that genome is chosen according to the following rules:

- Prefer phasings with more fully called alleles. For example, pick a phasing that results in a fully called allele and a no-called allele, rather than two alleles with no-calls.
- If two phasings result in the same number of fully called haplotypes, pick the phasing that results in the fewest number of fully called haplotypes within the population. For example, if the first genome has alleles ACCCCCT and TCCCCCCG, then for the second genome we prefer ACCCCCT and TCCCCCCG rather than two new alleles ACCCCCG and TCCCCCCT.

Computation of Calibrated Scores (CGA_CEHQ)

These qualities are derived by looking up the calibrated score using the varScoreEAF and totalReadCount of this locus in the Complete Genomics calibration. The false positive, undercall, and overcall calibrations are used. For more information, see *Complete Genomics Small Variant Score Calibration Methods*.

CNV Conversion

CNV records are mostly 1:1 transfers of values from the input CNV files, but there are several issues worth highlighting:

- The input files use 'N' to indicate various forms of missing or no-called data; these values are translated to period "." in the VCF file. This is also used in multi-sample VCFs of mixed gender, where coverage is no-called in regions of sex chromosomes absent from the given genome.
- While CNV data are reported in 2 kb windows, source non-diploid data are based on 100 kb windows. mkvcf translates this information by copying the information from a 100 kb window into every 2 kb window it contains. Due to the way the window boundaries are determined, a 2 kb window never spans the boundaries of two 100 kb windows.
- CNV records in the output represent the analysis windows rather than final CNV segments. This makes it easy to compare multiple genomes within a given window, but it requires that multiple rows be considered to identify the boundaries of a called segment.

MEI Conversion

MEI records are mostly 1:1 transfers of values from the input MEI files. However, the input files use "N" to indicate various forms of missing or no-called data; these values are translated to period "." in the VCF file.

SV Conversion

The "Specifying Complex Rearrangements with Breakends" section of the VCF 4.1 specifications describes the representation of complex structural variations as:

> An arbitrary rearrangement event can be summarized as a set of novel **adjacencies**. Each adjacency ties together 2 **breakends**. The two breakends at either end of a novel adjacency are called **mates.**

The specifications for structural variants in output from mkvcf follows the definitions and basic logic introduced in VCF 4.1. The concept of "junctions" in Complete Genomics data is translated to the concept of "adjacency" in the VCF format. Each adjacency ties together 2 breakends. The two breakends at either end of a novel adjacency are called mates. By extension, the left and right sections of a junction are analogous to mates:

- Adjacency: Analogous to Complete Genomics term "junction"
- Breakend: Analogous to Complete Genomics term '"LeftPosition" or "RightPosition"
- Mate: Analogous to Complete Genomics term "LeftSection" or "RightSection"

Hence, an individual SV record in the output corresponds to one side of a junction in the input file(s). Pairing of an individual record to that corresponding to the other end of the junction is also provided via MATEID tag. In the case of a single genome each junction results in two SV records, one per each junction side. In the case of two-genomes comparison, the clusters of compatible junctions (i.e. those located within the distance threshold from each other: see the `--junction-distance-tolerance` option) from different genomes are represented by one junction from each cluster.