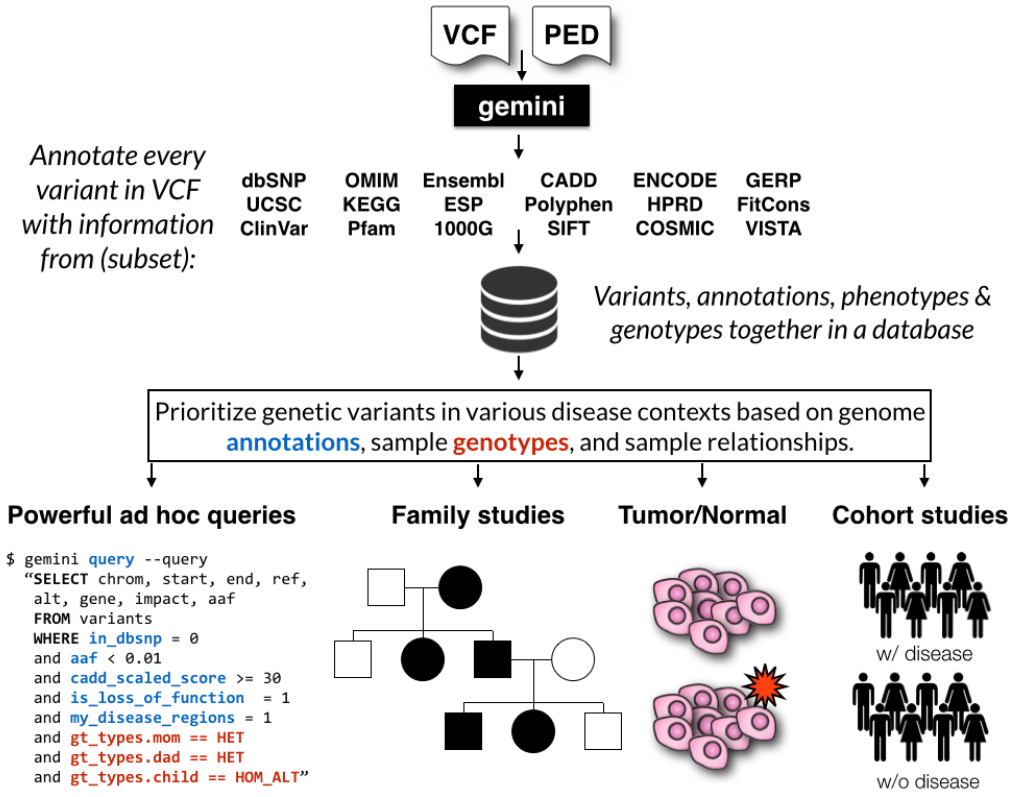
gemini Documentation

Release 0.20.1

Quinlan lab @ UVa

Sep 04, 2017

Contents


VCF
PED
gemini
Annotate every variant in VCF with information from (subset):
dbSNP
UCSC
ClinVar
OMIM
KEGG
Pfam
Ensembl
ESP
1000G
CADD
Polyphen
SIFT
ENCODE
HPRD
COSMIC
GERP
FitCons
VISTA
Variants, annotations, phenotypes & genotypes together in a database
Prioritize genetic variants in various disease contexts based on genome annotations, sample genotypes, and sample relationships.
Powerful ad hoc queries
Family studies
Tumor/Normal
Cohort studies
$ gemini query --query
"SELECT chrom, start, end, ref,
alt, gene, impact, aaf
FROM variants
WHERE in_dbsnp = 0
and aaf < 0.01
and cadd_scaled_score >= 30
and is_loss_of_function = 1
and my_disease_regions = 1
and gt_types.mom == HET
and gt_types.dad == HET
and gt_types.child == HOM_ALT"
w/ disease
w/o disease

CHAPTER 1

Overview

GEMINI (GEnome MINIng) is a flexible framework for exploring genetic variation in the context of the wealth of genome annotations available for the human genome. By placing genetic variants, sample phenotypes and genotypes, as well as genome annotations into an integrated database framework, `GEMINI` provides a simple, flexible, and powerful system for exploring genetic variation for disease and population genetics.

Using the GEMINI framework begins by loading a VCF file (and an optional PED file) into a database. Each variant is automatically annotated by comparing it to several genome annotations from source such as ENCODE tracks, UCSC tracks, OMIM, dbSNP, KEGG, and HPRD. All of this information is stored in portable SQLite database that allows one to explore and interpret both coding and non-coding variation using "off-the-shelf" tools or an enhanced SQL engine.

Please also see the original manuscript.

Note:

1. GEMINI solely supports human genetic variation mapped to build 37 (aka hg19) of the human genome.
2. GEMINI is very strict about adherence to VCF format 4.1.
3. For best performance, load and query GEMINI databases on the fastest hard drive to which you have access.

CHAPTER 2

Tutorials

In addition to the documentation, please review the following tutorials if you are new to GEMINI. We recommend that you follow these tutorials in order, as they introduce concepts that build upon one another.

- Introduction to GEMINI, basic variant querying and data exploration. html pdf
- Identifying de novo mutations underlying Mendelian disease html pdf
- Identifying autosomal recessive variants underlying Mendelian disease html pdf
- Identifying autosomal dominant variants underlying Mendelian disease html pdf
- Other GEMINI tools html pdf

CHAPTER 3

Latest news

New Installation

In version *0.18*, we have introduced a new installation procedure based on conda that should make the installation more reliable. For users with an existing installation with any trouble using *gemini update –devel*, we suggest to do a fresh install using a command like:

```
wget https://github.com/arq5x/gemini/raw/master/gemini/scripts/gemini_install.py
python gemini_install.py $tools $data
PATH=$tools/bin:$data/anaconda/bin:$PATH
```

where *$tools* and *$data* are paths writable on your system.

With an existing *$tool* and *$data* directory from a previous install, you can use the installer to re-install the Python code with the new version, but leave the existing data in place. To do this, first remove the old anaconda directory:

```
rm -rf $data/anaconda
```

then run the installation commands above.

Changes to Inheritance Tools

As of version 0.16.0, the built-in Mendelian inheritance tools are more stringent by default (they can be relaxed with the `--lenient`) option. By default, samples with unknown phenotype will not affect the tools, and strict requirements are placed on family structure. See the the *docs* for more info. In addition, the inheritance tools now support multi-generational pedigrees.

New GEMINI Workflow

As version 0.12.2 of GEMINI it is required that your input VCF file undergo additional preprocessing such that multi-allelic variants are decomposed and normalized using the vt toolset from the Abecasis lab. Note that we have also decomposed and normalized all of the VCF-based annotation files (e.g., ExAC, dbSNP, ClinVar, etc.) so that variants and alleles are properly annotated and we minimize false negative and false positive annotations. For a great discussion of why this is necessary, please read this blog post from Eric Minikel in Daniel MacArthur's lab.

Essentially, VCF preprocessing for GEMINI now boils down to the following steps.

0. If working with GATK VCFs, you need to correct the AD INFO tag definition to play nicely with *vt*.
1. Decompose the original VCF such that variants with multiple alleles are expanded into distinct variant records; one record for each REF/ALT combination.
2. Normalize the decomposed VCF so that variants are left aligned and represented using the most parsimonious alleles.
3. Annotate with VEP or snpEff.
4. bgzip and tabix.

A workflow for the above steps is given below.

```
# setup
VCF=/path/to/my.vcf
NORMVCF=/path/to/my.norm.vcf.gz
REF=/path/to/human.b37.fasta
SNPEFFJAR=/path/to/snpEff.jar

# decompose, normalize and annotate VCF with snpEff.
# NOTE: can also swap snpEff with VEP
zless $VCF \
   | sed 's/ID=AD,Number=./ID=AD,Number=R/' \
   | vt decompose -s - \
   | vt normalize -r $REF - \
   | java -Xmx4G -jar $SNPEFFJAR GRCh37.75 \
   | bgzip -c > $NORMVCF
tabix -p vcf $NORMVCF

# load the pre-processed VCF into GEMINI
gemini load --cores 3 -t snpEff -v $NORMVCF $db

# query away
gemini query -q "select chrom, start, end, ref, alt, (gts).(*) from variants" \
             --gt-filter "gt_types.mom == HET and \
                          gt_types.dad == HET and \
                          gt_types.kid == HOM_ALT" \
             $db
```

CHAPTER 4

Citation

If you use GEMINI in your research, please cite the following manuscript:

```
Paila U, Chapman BA, Kirchner R, Quinlan AR (2013)
GEMINI: Integrative Exploration of Genetic Variation and Genome Annotations.
PLoS Comput Biol 9(7): e1003153. doi:10.1371/journal.pcbi.1003153
```

CHAPTER 5

Table of contents

Installation

Automated installation

GEMINI contains an automated installation script which installs GEMINI along with required Python dependencies, third party software and data files.

```
$ wget https://raw.github.com/arq5x/gemini/master/gemini/scripts/gemini_install.py
```

While other tools (e.g., `curl`) can be used to download this file, `wget` is required by the installation script. Please install `wget` if it is not already available:

Once you have downloaded the above installation script, you can proceed as follows:

```
$ python gemini_install.py /usr/local /usr/local/share/gemini
$ export PATH=$PATH:/usr/local/gemini/bin
# it would be wise to add the above line to your ``.bashrc`` or ``.bash_profile``
```

This installs the GEMINI executable as `/usr/local/bin/gemini`, other required third party dependencies in `/usr/local/bin`, and associated data files in `/usr/local/share/gemini`. **Please note that this is merely an example: one can place the GEMINI executables and annotation files in any directories one wishes.**

Note: The automated installtion script typically requires ~30 minutes, primarily owing to the time required to download the GEMINI genome annotation files. Also, please note that the annotation files **requires ~15Gb of storage**, so please ensure that the data directory (`/usr/local/share/gemini` in the example above) has sufficient space.

Tip: Some tips and tricks for installation issues:

1. Some older versions of wget have certificate problems with GitHub files. If you run into this problem, you can alternatively download the install script using‘‘wget –no-check-certificates‘‘ or `curl -O`.

2. The installation script is idempotent and you can re-run it multiple times without any issues. If you experience internet connectivity or other transient errors during installation, a re-run can often solve the problem (fingers crossed).
3. If you're installing behind a proxy you'll need to set proxy information in a condarc file and also set `all_proxy`, `http_proxy` and `https_proxy` in your `.bashrc` file. See this mailing list discussion for more information.
4. The installer tries to ignore site-wide environmental variables pointing to other Python installations, but if you have issues with pulling in Python libraries from other locations, you clear these by unsetting/setting these environmental variables before running: `unset PYTHONPATH && unset PYTHONHOME && export PYTHONNOUSERSITE=1`

Dependencies

The installer requires:

- Python 2.7.x
- git
- wget
- a working C / C++ compiler such as gcc
- zlib (including headers)

These dependencies already exist on most UNIX/LINUX/OS X machines. However, on minimalist machines **such as fresh Amazon Cloud instances**, one may have to install these dependencies before running the automated installer. In the case of Amazon instances, the following command should take care of all of the above dependencies:

```
$ sudo yum -y install python27 git wget gcc gcc-c++ zlib-devel
```

Installing without root access.

As many users do not have root or sudo access, the automated installer also has options to install in "non-root" environments:

```
$ python gemini_install.py ~/gemini ~/gemini --nosudo
```

Updating your PATH to find the GEMINI executable

At this point, you will have a self-contained installation of GEMINI, including both the software and its associated genome annotations. However, if you have done a custom install in a "non-root" enviornment, you will first need to update your `PATH` environment variable to include the path to the bin directory that you just created by running the automated installer.

For example, if, as above, you placed you custom install in `~/gemini`, you would need to update your `PATH` as follows. It would be wise to also add this to your `.bashrc` or `.bash_profile`:

```
$ export PATH=$PATH:~/gemini/bin
```

Note that this change will only last for the life of your current terminal session. To make this more permanent, update your `.bash_profile` so that this change is made each time you login.

If successful, you should be able to run the following command from anywhere on your system:

```
$ gemini -v
gemini 0.3.0b
```

Running unit tests

gemini only installs the software itself, not the source repository with the tests. To run the tests, do:

```
$ git clone https://github.com/arq5x/gemini
$ git checkout v0.18.3 # or the current version that you installed
$ cd gemini
$ bash master-test.sh
```

Updating the GEMINI executables and annotations

Once installed with the automated installer, it is easy to upgrade the GEMINI programs and associated data files to the latest released version with:

```
$ gemini update
```

There are also flags to update to the latest development version of the code or to only update data files without updating the source:

```
$ gemini update --devel
$ gemini update --dataonly
```

To download optional large files associated with specific analyses in GEMINI, like GERP scores per base pair and CADD scores, pass the `--extra` flag:

```
$ gemini update --dataonly --extra cadd_score
$ gemini update --dataonly --extra gerp_bp
```

Software dependencies

GEMINI depends upon several widely-used genomics command line software as well as multiple Python packages. We recognize that the dependency stack is quite deep and are working on ways to minimize dependencies in the interest of the most streamlined installation process possible. Nonetheless, the following are core dependencies:

1. Python 2.7.x
2. grabix
3. tabix (only by annotate tool)
4. bedtools (only by windower tool)
5. pybedtools (only by windower tool)

Manual installation

Once the above dependencies have been installed, one can begin installing `GEMINI` itself. To install you should download the latest source code from GitHub, either by going to:

```
http://github.com/arq5x/gemini
```

and clicking on "Downloads", or by cloning the git repository with:

```
$ git clone https://github.com/arq5x/gemini.git
```

Once you have the source code, run:

```
$ cd gemini
$ sudo python setup.py install
```

to install it. If you don't have permission to install it in the default directory, you can simply build the source in-place and use the package from the git repository:

```
$ python setup.py build_ext --inplace
```

Installing annotation files

One of the more appealing features in `GEMINI` is that it automatically annotates variants in a VCF file with several genome annotations. However, you must first install these data files on your system. It's easy enough — you just need to run the following script and tell it in which full path you'd like to install the necessary data files. The recommended path is `/usr/local/share`, but you can install the data files wherever you want.

```
$ python gemini/install-data.py /usr/local/share/
```

Note: Annotation files like GERP at base pair resolution and CADD scores are not part of this default installation owing to their large file size. They may however be installed as additional data files using the `gemini update --dataonly` option, with the flag `--extra` for `gerp_bp` and `cadd_score`.

Using previously installed annotation files

If you have installed GEMINI with the annotation files on a server and you can NFS mount the annotation files, you can tell a local install of GEMINI where those annotation files are by making the file ~/.gemini/gemini-config.yaml:

```
annotation_dir: /path/to/nfs_mounted/gemini/data
versions:
  GRCh37-gms-mappability.vcf.gz: 2
  hg19.rmsk.bed.gz: 2
```

Running the testing suite

GEMINI comes with a full test suite to make sure that everything has installed correctly on your system. We **strongly** encourage you to run these tests.

```
$ bash master-test.sh
```

Quick start

gemini is designed to allow researchers to explore genetic variation contained in a VCF file. The basic workflow for working with *gemini* is outlined below.

Importing VCF files into gemini.

Note: we now recommend splitting variants with multiple alternates and left-aligning, and trimming all variants before loading into gemini.

See *Step 1. split, left-align, and trim variants* for a detailed explanation.

Before we can use GEMINI to explore genetic variation, we must first `load` our VCF file into the GEMINI database framework. We expect you to have first annotated the functional consequence of each variant in your VCF using either VEP or snpEff (Note that v3.0+ of snpEff is required to track the amino acid length of each impacted transcript). Logically,the loading step is done with the `gemini load` command. Below are two examples based on a VCF file that we creatively name my.vcf. The first example assumes that the VCF has been pre-annotated with VEP and the second assumes snpEff.

```
# VEP-annotated VCF
$ gemini load -v my.vcf -t VEP my.db

# snpEff-annotated VCF
$ gemini load -v my.vcf -t snpEff my.db
```

Assuming you have a valid VCF file produced by standard variation discovery programs (e.g., GATK, FreeBayes, etc.), one loads the VCF into the gemini framework with the **load** submodule:

```
$ gemini load -v my.vcf my.db
```

In this step, *gemini* reads and loads the my.vcf file into a SQLite database named my.db, whose structure is described here. While loading the database, *gemini* computes many additional population genetics statistics that support downstream analyses. It also stores the genotypes for each sample at each variant in an efficient data structure that minimizes the database size.

Loading is by far the slowest aspect of GEMINI. Using multiple CPUs can greatly speed up this process.

```
$ gemini load -v my.vcf --cores 8 my.db
```

Querying the *gemini* database.

If you are familiar with SQL, `gemini` allows you to directly query the database in search of interesting variants via the *-q* option. For example, here is a query to identify all novel, loss-of-function variants in your database:

```
$ gemini query -q "select * from variants where is_lof = 1 and in_dbsnp = 0" my.db
```

Or, we can ask for all variants that substantially deviate from Hardy-Weinberg equilibrium:

```
$ gemini query -q "select * from variants where hwe < 0.01" my.db
```

Annotation with snpEff or VEP

GEMINI depends upon external tools to predict the functional consequence of variants in a VCF file. We currently support annotations produced by either SnpEff or VEP.

Note: Versions tested: VEP versions 73 through 75 and core SnpEff versions 3.0 through 3.6. GEMINI supports ENSEMBL annotations hence users are expected to download genome databases for these tools as represented in the examples below.

Note: Version support would be subsequently updated here, as we test along and add or edit changes available with the latest version of these tools.

Recommended instructions for annotating existing VCF files with these tools are summarized here.

Stepwise installation and usage of VEP

Download the Variant Effect Predictor "standalone perl script" from Ensembl. You can choose a specific version of VEP to download here

Example:

```
Download version 74
```

Untar the tarball into the current directory

```
$ tar -zxvf variant_effect_predictor.tar.gz
```

This will create the variant_effect_predictor directory. Now do the following for install:

```
$ cd variant_effect_predictor
$ perl INSTALL.pl [options]
```

By default this would install the API's, bioperl-1.2.3 and the cache files (in the $HOME/.vep directory).

Homebrew or Anaconda VEP installation

If you are a Homebrew, Linuxbrew or Anaconda user, there is an automated recipe to install the main VEP script and plugins in the CloudBioLinux homebrew repository:

```
$ brew tap chapmanb/cbl
$ brew update
$ brew install vep
```

For Anaconda/Miniconda, just make sure you are pointing to the **bioconda** channel:

```
$ conda install variant-effect-predictor -c bioconda
```

Manual installation of VEP

For those (e.g mac users) who have a problem installing through this install script, try a manual installation of the API's, BioPerl-1.2.3 and set all pre-requisites for running VEP (`DBI` and `DBD::mysql` modules required). The appropriate pre-build caches should be downloaded for Human to the `$HOME/.vep` directory and then untar.

You may follow instructions at http://www.ensembl.org/info/docs/api/api_installation.html which provides alternate options for the API installation and additional tips for windows/mac users. It also has information for setting up your environment to run VEP.

Example download of the cache files

```
$ wget ftp://ftp.ensembl.org/pub/release-73/variation/VEP/homo_sapiens_vep_73.tar.gz
```

You may change the release date in this example to get the appropriate cache files for your version of VEP that you have installed.

Example

```
$ wget ftp://ftp.ensembl.org/pub/release-74/variation/VEP/homo_sapiens_vep_74.tar.gz
```

Cache requires the `gzip` and `zcat` utilities. VEP uses `zcat` to decompress cached files. For systems where zcat may not be installed or may not work, the following option needs to be added along with the `--cache` option:

```
--compress "gunzip -c"
```

Running VEP

You may now run VEP as:

```
$ perl variant_effect_predictor.pl [OPTIONS]
```

We recommend running VEP with the following options as currently we support VEP fields specified as below:

```
$ perl variant_effect_predictor.pl -i example.vcf \
    --cache \
    --sift b \
    --polyphen b \
    --symbol \
    --numbers \
    --biotype \
    --total_length \
    -o output \
    --vcf \
    --fields Consequence,Codons,Amino_acids,Gene,SYMBOL,Feature,EXON,PolyPhen,SIFT,
↪Protein_position,BIOTYPE
```

A documentation for the above specified options may be found at http://www.ensembl.org/info/docs/tools/vep/script/vep_options.html

As of GEMINI version 0.8.0, you can also run VEP with additional fields, which will be automatically added to the variants table as columns. As an example, run VEP on your VCF with the dbNSFP and LOFTEE plugins to annotate potential high impact variations:

```
$ variant_effect_predictor.pl --sift b --polyphen b --symbol --numbers --biotype \
--total_length --canonical --ccds \
--fields Consequence,Codons,Amino_acids,Gene,SYMBOL,Feature,EXON,PolyPhen,SIFT,
↪Protein_position,BIOTYPE,CANONICAL,CCDS,RadialSVM_score,RadialSVM_pred,LR_score,LR_
↪pred,CADD_raw,CADD_phred,Reliability_index,LoF,LoF_filter,LoF_flags \
```

```
--plugin dbNSFP,/path/to/dbNSFP_v2.5.gz,RadialSVM_score,RadialSVM_pred,LR_score,LR_
↪pred,CADD_raw,CADD_phred,Reliability_index \
--plugin LoF,human_ancestor_fa:/path/to/human_ancestor.fa
```

Feeding this into GEMINI produces a variants table with columns for each of the additional VEP metrics. The annotation loader names each column by prefixing `vep_` to the origin VEP name, so select on `vep_radialsvm_score` or `vep_lof_filter` in the final database.

Stepwise installation and usage of SnpEff

Note: Basic Requirements: Java v1.7 or later; at least 4GB of memory

Download the supported versions of SnpEff from http://snpeff.sourceforge.net/download.html

Example:

```
$ wget http://sourceforge.net/projects/snpeff/files/snpEff_v3_6_core.zip
```

Note: SnpEff should be installed preferably in `snpEff` directory in your home directory. Else, you must update the `data_dir` parameter in your snpEff.config file. For e.g. if the installation of snpEff has been done in `~/src` instead of `~/` then change the data_dir parameter in snpEff.config to `data_dir = ~/src/snpEff/data/`

Unzip the downloaded package.

```
$ unzip snpEff_v3_6_core.zip
```

Change to the `snpEff` directory and download the genome database.

```
$ cd snpEff_v3_6_core
$ java -jar snpEff.jar download GRCh37.69
```

Unzip the downloaded genome database. This will create and place the genome in the 'data' directory

```
$ unzip snpEff_v3_6_GRCh37.69.zip
```

To annotate a vcf using snpEff, use the `default` options as below:

Note: Memory options for the run may be specified as `-Xmx4G` (4GB)

```
$ java -Xmx4G -jar snpEff.jar -i vcf -o vcf GRCh37.69 example.vcf > example_snpeff.vcf
```

If running from a directory different from the installation directory, the complete path needs to be specified as, e.g.:

```
$ java -Xmx4G -jar path/to/snpEff/snpEff.jar -c path/to/snpEff/snpEff.config GRCh37.
↪69 path/to/example.vcf > example_snpeff.vcf
```

Note: When using the latest versions of snpEff (e.g. 4.1) annotate your VCF with the additional parameters `-classic` and `-formatEff`. This would ensure proper loading of the gene info columns in the variants table.

Columns populated by snpEff/VEP tools

The following variant consequence columns in the variant/variant_impacts table, are populated with these annotations, which are otherwise set to null.

- anno_id
- gene
- transcript
- exon
- is_exonic
- is_lof
- is_coding
- codon_change
- aa_change
- aa_length
- biotype
- impact
- impact_so
- impact_severity
- polyphen_pred
- polyphen_score
- sift_pred
- sift_score

Standardizing `impact` definitions for GEMINI

GEMINI uses slightly modified impact terms (for ease) to describe the functional consequence of a given variant as provided by snpEff/VEP.

The table below shows the alternate *GEMINI* terms used for *snpEff/VEP*.

GEMINI terms	snpEff terms	VEP terms (uses SO by default)
splice_acceptor	SPLICE_SITE_ACCEPTOR	splice_acceptor_variant
splice_donor	SPLICE_SITE_DONOR	splice_donor_variant
stop_gain	STOP_GAINED	stop_gained
stop_loss	STOP_LOST	stop_lost
frame_shift	FRAME_SHIFT	frameshift_variant
start_loss	START_LOST	null
exon_deleted	EXON_DELETED	null
non_synonymous_start	NON_SYNONYMOUS_START	null
transcript_codon_change	null	initiator_codon_variant
chrom_large_del	CHROMOSOME_LARGE_DELETION	null
rare_amino_acid	RARE_AMINO_ACID	null
non_syn_coding	NON_SYNONYMOUS_CODING	missense_variant

Table 5.1 – continued from previous page

GEMINI terms	snpEff terms	VEP terms (uses SO by default)
inframe_codon_gain	CODON_INSERTION	inframe_insertion
inframe_codon_loss	CODON_DELETION	inframe_deletion
inframe_codon_change	CODON_CHANGE	null
codon_change_del	CODON_CHANGE_PLUS_CODON_DELETION	null
codon_change_ins	CODON_CHANGE_PLUS_CODON_INSERTION	null
UTR_5_del	UTR_5_DELETED	null
UTR_3_del	UTR_3_DELETED	null
splice_region	SPLICE_SITE_REGION	splice_region_variant
mature_miRNA	null	mature_miRNA_variant
regulatory_region	null	regulatory_region_variant
TF_binding_site	null	TF_binding_site_variant
regulatory_region_ablation	null	regulatory_region_ablation
regulatory_region_amplification	null	regulatory_region_amplification
TFBS_ablation	null	TFBS_ablation
TFBS_amplification	null	TFBS_amplification
synonymous_stop	SYNONYMOUS_STOP	stop_retained_variant
synonymous_coding	SYNONYMOUS_CODING	synonymous_variant
UTR_5_prime	UTR_5_PRIME	5_prime_UTR_variant
UTR_3_prime	UTR_3_PRIME	3_prime_UTR_variant
intron	INTRON	intron_variant
CDS	CDS	coding_sequence_variant
upstream	UPSTREAM	upstream_gene_variant
downstream	DOWNSTREAM	downstream_gene_variant
intergenic	INTERGENIC	intergenic_variant
intergenic_conserved	INTERGENIC_CONSERVED	null
intragenic	INTRAGENIC	null
gene	GENE	null
transcript	TRANSCRIPT	null
exon	EXON	null
start_gain	START_GAINED	null
synonymous_start	SYNONYMOUS_START	null
intron_conserved	INTRON_CONSERVED	null
nc_transcript	null	nc_transcript_variant (should have been ret
NMD_transcript	null	NMD_transcript_variant
incomplete_terminal_codon	null	incomplete_terminal_codon_variant
nc_exon	null	non_coding_exon_variant (should have bee
transcript_ablation	null	transcript_ablation
transcript_amplification	null	transcript_amplification
feature elongation	null	feature_elongation
feature truncation	null	feature_truncation

Note: "null" refers to the absence of the corresponding term in the alternate database

SO `impact` definitions in GEMINI

The below table shows the *Sequence Ontology (SO)* term mappings for the GEMINI impacts, which is otherwise contained in the `impact_so` column of the variants/variant_impacts table of the GEMINI database. The last column shows the severity terms defined in GEMINI for these impacts.

GEMINI terms (column: impact)	Sequence Ontology terms (column: impact_so)	Impact severity
splice_acceptor	splice_acceptor_variant	HIGH
splice_donor	splice_donor_variant	HIGH
stop_gain	stop_gained	HIGH
stop_loss	stop_lost	HIGH
frame_shift	frameshift_variant	HIGH
start_loss	start_lost	HIGH
exon_deleted	exon_loss_variant	HIGH
non_synonymous_start	initiator_codon_variant	HIGH
transcript_codon_change	initiator_codon_variant	HIGH
chrom_large_del	chromosomal_deletion	HIGH
rare_amino_acid	rare_amino_acid_variant	HIGH
non_syn_coding	missense_variant	MED
inframe_codon_gain	inframe_insertion	MED
inframe_codon_loss	inframe_deletion	MED
inframe_codon_change	coding_sequence_variant	MED
codon_change_del	disruptive_inframe_deletion	MED
codon_change_ins	disruptive_inframe_insertion	MED
UTR_5_del	5_prime_UTR_truncation + exon_loss_variant	MED
UTR_3_del	3_prime_UTR_truncation + exon_loss_variant	MED
splice_region	splice_region_variant	MED
mature_miRNA	mature_miRNA_variant	MED
regulatory_region	regulatory_region_variant	MED
TF_binding_site	TF_binding_site_variant	MED
regulatory_region_ablation	regulatory_region_ablation	MED
regulatory_region_amplification	regulatory_region_amplification	MED
TFBS_ablation	TFBS_ablation	MED
TFBS_amplification	TFBS_amplification	MED
synonymous_stop	stop_retained_variant	LOW
synonymous_coding	synonymous_variant	LOW
UTR_5_prime	5_prime_UTR_variant	LOW
UTR_3_prime	3_prime_UTR_variant	LOW
intron	intron_variant	LOW
CDS	coding_sequence_variant	LOW
upstream	upstream_gene_variant	LOW
downstream	downstream_gene_variant	LOW
intergenic	intergenic_variant	LOW
intergenic_conserved	conserved_intergenic_variant	LOW
intragenic	intragenic_variant	LOW
gene	gene_variant	LOW
transcript	transcript_variant	LOW
exon	exon_variant	LOW
start_gain	5_prime_UTR_premature_start_codon_gain_variant	LOW
synonymous_start	start_retained_variant	LOW
intron_conserved	conserved_intron_variant	LOW
nc_transcript	nc_transcript_variant	LOW
NMD_transcript	NMD_transcript_variant	LOW
incomplete_terminal_codon	incomplete_terminal_codon_variant	LOW
nc_exon	non_coding_exon_variant	LOW
transcript_ablation	transcript_ablation	LOW
transcript_amplification	transcript_amplification	LOW

Continued on next page

Table 5.2 – continued from previous page

GEMINI terms (column: impact)	Sequence Ontology terms (column: impact_so)	Impact severity
feature elongation	feature_elongation	LOW
feature truncation	feature_truncation	LOW

Preprocessing and Loading a VCF file into GEMINI

Step 1. split, left-align, and trim variants

Variants with multiple alternate alleles will not be handled correctly by gemini (or by the tools used to annotate the variants). As projects get more samples it is likely that a non-negligible percentage of site will have multiple alternate alleles.

In addition, variants that are not left-aligned and trimmed can be incorrectly (or not) annotated.

To reduce the number of false negatives, **we strongly recommend that gemini users split, left-align, and trim their variants**. The tools we recommend for this are either vt:

```
vt decompose -s $VCF | vt normalize -r $REFERENCE - > $NEW_VCF
```

gemini uses the allele depths from the AD tag. In order for *vt* to decompose correctly, users will have to change the #INFO field for AD in the header from Number=. to Number=R.

Then the *$NEW_VCF* can be annotated with snpEff or VEP.

Step 2. Annotate with snpEff or VEP

Note: Annotate your VCF with SnpEff/VEP, prior to loading it into GEMINI, otherwise the gene/transcript features would be set to None.

GEMINI supports gene/transcript level annotations (we do not use pre-computed values here) from snpEff and VEP and hence we suggest that you first annotate your VCF with either of these tools, prior to loading it into GEMINI. The related database columns would be populated, which would otherwise be set to None if an unannotated VCF file is loaded into GEMINI.

Note: Choose the annotator as per your requirement! Some gene/transcript annotations are available with only one tool (e.g. `Polyphen/Sift` with VEP). As such these values would be set to None, if an alternate annotator is used during the load step.

Instructions for installing and running these tools can be found in the following section:

Annotation with snpEff or VEP

The basics

Before we can use GEMINI to explore genetic variation, we must first `load` our VCF file into the GEMINI database framework. We expect you to have first annotated the functional consequence of each variant in your VCF using either VEP or snpEff (Note that v3.0+ of snpEff is required to track the amino acid length of each impacted transcript). Logically, the loading step is done with the `gemini load` command. Below are two examples based on a VCF file

that we creatively name my.vcf. The first example assumes that the VCF has been pre-annotated with VEP and the second assumes snpEff.

```
# VEP-annotated VCF
$ gemini load -v my.vcf -t VEP my.db

# snpEff-annotated VCF
$ gemini load -v my.vcf -t snpEff my.db
```

As each variant is loaded into the GEMINI database framework, it is being compared against several annotation files that come installed with the software. We have developed an annotation framework that leverages tabix, bedtools, and pybedtools to make things easy and fairly performant. The idea is that, by augmenting VCF files with many informative annotations, and converting the information into a sqlite database framework, GEMINI provides a flexible database-driven API for data exploration, visualization, population genomics and medical genomics. We feel that this ability to integrate variation with the growing wealth of genome annotations is the most compelling aspect of GEMINI. Combining this with the ability to explore data with SQL using a database design that can scale to 1000s of individuals (genotypes too!) makes for a nice, standardized data exploration system.

Many variant callers set filter flags in the VCF file to flag possible problem variants. By default GEMINI will leave these variants in the database during loading but they can be filtered out during the loading step by passing the --passonly flag to load.

You can create a smaller, faster database if you dont need the genotype likelihoods format each sample by passing the *–skip-pls* flag.

Using multiple CPUs for loading

Now, the loading step is very computationally intensive and thus can be very slow with just a single core. However, if you have more CPUs in your arsenal, you can specify more cores. This provides a roughly linear increase in speed as a function of the number of cores. On our local machine, we are able to load a VCF file derived from the exomes of 60 samples in about 10 minutes. With a single core, it takes a few hours.

Note: Using multiple cores requires that you have both the bgzip tool from tabix and the grabix tool installed in your PATH.

```
$ gemini load -v my.vcf -t snpEff --cores 20 my.db
```

Using LSF, SGE, SLURM and Torque schedulers

One can load VCF files into GEMINI in parallel using many cores on LSF, SGE, SLURM or Torque clusters. One must simply specify the type of job scheduler your cluster uses and the queue name to which your jobs should be submitted.

For example, let's assume you use LSF and a queue named preempt_everyone. Here is all you need to do:

```
$ gemini load -v my.vcf \
         -t snpEff \
         --cores 50 \
         --queue preempt_everyone \
         --scheduler lsf \
         my.db
```

Describing samples with a PED file

GEMINI also accepts PED files in order to establish the familial relationships and phenotypic information of the samples in the VCF file.

```
$ gemini load -v my.vcf -p my.ped -t snpEff my.db
```

The PED file format is documented here: PED. An example PED file looks like this:

1 M10475 -9 -9 1 1
1 M10478 M10475 M10500 2 2
1 M10500 -9 -9 2 2
1 M128215 M10475 M10500 1 1

The columns are `family_id`, `name`, `paternal_id`, `maternal_id`, `sex` and `phenotype`. For GEMINI, you can use either tabs or spaces, but not both.

You can also provide a PED file with a heading starting with #, and include extra fields, like this:

#family_id name paternal_id maternal_id sex phenotype hair_color
1 M10475 -9 -9 1 1 brown
1 M10478 M10475 M10500 2 2 brown
1 M10500 -9 -9 2 2 black
1 M128215 M10475 M10500 1 1 blue

This will add the extra columns to the `samples` table and allow for you to use those extra columns during queries.

Missing values for `family_id`, `paternal_id` and `maternal_id` can be specified by any of `0`, `-9` or `None` and GEMINI will translate them to `0` in the database.

Load GERP base pair conservation scores

GERP scores at base pair resolution are loaded by default (Note: This requires a prior install of the data file by running `gemini update --dataonly --extra gerp_bp`). However, if not required, one may optionally skip the load process (to save on the loading time) with the `--skip-gerp-bp` option.

```
$ gemini load -v my.vcf --skip-gerp-bp -t snpEff my.db
```

Load CADD scores for deleterious variants

CADD scores (http://cadd.gs.washington.edu/) are loaded by default in GEMINI (Note: This requires a prior install of the data file by running `gemini update --dataonly --extra cadd_score`). However, one may optionally skip the load process using the `--skip-cadd` option.

```
$ gemini load -v my.vcf --skip-cadd my.db
```

Updating the samples table in a database

If, after loading a database, you find more information about your samples or want to add a column to the samples table to query on, you can reload the samples table with a new PED file with `gemini amend --sample`. This is also useful if you forgot to load a PED file when initially loading your database. This file must have the standard first six columns of a PED file, but after that other columns can be added. The top of the PED file also must have a header starting with # which names all of the columns if there are more than the standard six PED file columns:

```
$ gemini amend --sample your_new_ped_file your.db
```

Loading VCFs without genotypes.

To do.

Querying the GEMINI database

The real power in the `GEMINI` framework lies in the fact that all of your genetic variants have been stored in a convenient database in the context of a wealth of genome annotations that facilitate variant interpretation. The expressive power of SQL allows one to pose intricate questions of one's variation data.

Note: If you are unfamiliar with SQL, sqlzoo has a decent online tutorial describing the basics. Really all you need to learn is the SELECT statement, and the examples below will give you a flavor of how to compose base SQL queries against the GEMINI framework.

Basic queries

GEMINI has a specific tool for querying a gemini database that has been loaded using the `gemini load` command. That's right, the tool is called `gemini query`. Below are a few basic queries that give you a sense of how to interact with the gemini database using the `query` tool.

1. Extract all transitions with a call rate > 95%

```
$ gemini query -q "select * from variants \
                   where sub_type = 'ts' \
                   and call_rate >= 0.95" my.db
```

2. Extract all loss-of-function variants with an alternate allele frequency < 1%:

```
$ gemini query -q "select * from variants \
                   where is_lof = 1 \
                   and aaf >= 0.01" my.db
```

3. Extract the nucleotide diversity for each variant:

```
$ gemini query -q "select chrom, start, end, pi from variants" my.db
```

4. Combine `GEMINI` with `bedtools` to compute nucleotide diversity estimates across 100kb windows:

```
$ gemini query -q "select chrom, start, end, pi from variants \
                   order by chrom, start, end" my.db | \
  bedtools map -a hg19.windows.bed -b - -c 4 -o mean
```

Selecting sample genotypes

The above examples illustrate *ad hoc* queries that do not request or filter upon the genotypes of individual samples. Since GEMINI stores the genotype information for each variant in compressed arrays that are stored as BLOBs in the database, standard SQL queries cannot directly access individual genotypes. However, we have enhanced the SQL syntax to support such queries with C "struct-like" access. For example, to retrieve the alleles for a given sample's (in this case, sample 1094PC0009), one would add `gts.1094PC0009` to the select statement.

Here is an example of selecting the genotype alleles for four different samples (note the examples below use the test.snpEff.vcf.db file that is created in the ./test directory when you run the *bash master-test.sh* command as described above):

```
$ gemini query -q "select chrom, start, end, ref, alt, gene, \
                          gts.1094PC0005, \
                          gts.1094PC0009, \
                          gts.1094PC0012, \
                          gts.1094PC0013 \
                   from variants" test.snpEff.vcf.db

chr1        30547   30548   T       G       FAM138A ./.     ./.     ./.     ./.
chr1        30859   30860   G       C       FAM138A G/G     G/G     G/G     G/G
chr1        30866   30869   CCT     C       FAM138A CCT/CCT CCT/CCT CCT/C   CCT/CCT
chr1        30894   30895   T       C       FAM138A T/C     T/C     T/T     T/T
chr1        30922   30923   G       T       FAM138A ./.     ./.     ./.     ./.
chr1        69269   69270   A       G       OR4F5   ./.     ./.     G/G     G/G
chr1        69427   69428   T       G       OR4F5   T/T     T/T     T/T     T/T
chr1        69510   69511   A       G       OR4F5   ./.     ./.     A/G     A/G
chr1        69760   69761   A       T       OR4F5   A/A     A/T     A/A     A/A
chr1        69870   69871   G       A       OR4F5   ./.     G/G     G/G     G/G
```

You can also add a header so that you can keep track of who's who:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene, \
                          gts.1094PC0005, \
                          gts.1094PC0009, \
                          gts.1094PC0012, \
                          gts.1094PC0013 \
                   from variants" \
                   --header test.snpEff.vcf.db

chrom       start   end     ref     alt     gene gts.1094PC0005     gts.1094PC0009 ␣
↪gts.1094PC0012  gts.1094PC0013
chr1        30547   30548   T       G       FAM138A ./.     ./.     ./.     ./.
chr1        30859   30860   G       C       FAM138A G/G     G/G     G/G     G/G
chr1        30866   30869   CCT     C       FAM138A CCT/CCT CCT/CCT CCT/C   CCT/CCT
chr1        30894   30895   T       C       FAM138A T/C     T/C     T/T     T/T
chr1        30922   30923   G       T       FAM138A ./.     ./.     ./.     ./.
chr1        69269   69270   A       G       OR4F5   ./.     ./.     G/G     G/G
chr1        69427   69428   T       G       OR4F5   T/T     T/T     T/T     T/T
chr1        69510   69511   A       G       OR4F5   ./.     ./.     A/G     A/G
chr1        69760   69761   A       T       OR4F5   A/A     A/T     A/A     A/A
chr1        69870   69871   G       A       OR4F5   ./.     G/G     G/G     G/G
```

Let's now get the genotype and the depth of aligned sequence observed for a sample so that we can assess the confidence in the genotype:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene,
                      gts.1094PC0005, \
```

```
                    gt_depths.1094PC0005 \
                from variants" test.snpEff.vcf.db

chr1    30547   30548   T       G       FAM138A ./.     -1
chr1    30859   30860   G       C       FAM138A G/G     7
chr1    30866   30869   CCT     C       FAM138A CCT/CCT 8
chr1    30894   30895   T       C       FAM138A T/C     8
chr1    30922   30923   G       T       FAM138A ./.     -1
chr1    69269   69270   A       G       OR4F5   ./.     -1
chr1    69427   69428   T       G       OR4F5   T/T     2
chr1    69510   69511   A       G       OR4F5   ./.     -1
chr1    69760   69761   A       T       OR4F5   A/A     1
chr1    69870   69871   G       A       OR4F5   ./.     -1
```

Selecting sample genotypes based on "wildcards".

The above examples demonstrate how one can select individual sample genotype information by explicitly listing each column and sample that one wishes to see. Obviously, this can become tedious when a project involves hundreds or thousands of samples — if you wanted to see genotype information for the 345 of 1145 affected samples in your study, you would have to type each and every column.sample name out. Brutal.

The "*" wildcard

To get around this, one can bulk-select sample genotype information using "wildcards". The column and the wildcard must each be surrounded with parentheses and separated by a period. The "*" is a shortcut (wildcard) meaning "all samples".

Note: The syntax for SELECTing genotype columns using a wildcard is `(COLUMN).(WILDCARD).`

For example, a shortcut to reporting the genotype for *all* samples (in this case 4) in the study, one could do the following:

```
$ gemini query --header -q "select chrom, start, end, ref, alt, gene, (gts).(*) \
                            from variants" extended_ped.db
chrom start end ref alt gene  gts.M10475  gts.M10478  gts.M10500  gts.M128215
chr10 1142207 1142208 T C WDR37 C/C C/C C/C C/C
chr10 48003991  48003992  C T ASAH2C  T/T C/T C/T C/C
chr10 52004314  52004315  T C ASAH2 ./. ./. C/C C/C
chr10 52497528  52497529  G C ASAH2B  ./. C/C C/C ./.
chr10 126678091 126678092 G A CTBP2 G/G G/G G/G G/A
chr10 135210790 135210791 T C MTG1.1  T/T C/C C/C T/T
chr10 135336655 135336656 G A SPRN  ./. A/A ./. A/A
chr10 135369531 135369532 T C SYCE1 T/T T/C T/C T/T
chr16 72057434  72057435  C T DHODH C/T C/C C/C C/C
```

Wildcards based on sample attributes

To report the genotypes for solely those samples that are affected (phenotype == 2) with the phenotype in question, one could do the following:

```
$ gemini query --header -q "select chrom, start, end, ref, alt, gene, (gts).
↪(phenotype==2) \
                           from variants" extended_ped.db
chrom start end ref alt gene  gts.M10478  gts.M10500
chr10 1142207 1142208 T C WDR37 C/C C/C
chr10 48003991  48003992  C T ASAH2C  C/T C/T
chr10 52004314  52004315  T C ASAH2 ./. C/C
chr10 52497528  52497529  G C ASAH2B  C/C C/C
chr10 126678091 126678092 G A CTBP2 G/G G/G
chr10 135210790 135210791 T C MTG1.1  C/C C/C
chr10 135336655 135336656 G A SPRN  A/A ./.
chr10 135369531 135369532 T C SYCE1 T/C T/C
chr16 72057434  72057435  C T DHODH C/C C/C
```

One can add multiple wildcard criteria as well:

```
$ gemini query --header -q "select chrom, start, end, ref, alt, gene, (gts).
↪(phenotype==1 and hair_color=='blue') \
                           from variants" extended_ped.db
chrom start end ref alt gene  gts.M128215
chr10 1142207 1142208 T C WDR37 C/C
chr10 48003991  48003992  C T ASAH2C  C/C
chr10 52004314  52004315  T C ASAH2 C/C
chr10 52497528  52497529  G C ASAH2B  ./.
chr10 126678091 126678092 G A CTBP2 G/A
chr10 135210790 135210791 T C MTG1.1  T/T
chr10 135336655 135336656 G A SPRN  A/A
chr10 135369531 135369532 T C SYCE1 T/T
chr16 72057434  72057435  C T DHODH C/C
```

`--gt-filter` Filtering on genotypes

Now, we often want to focus only on variants where a given sample has a specific genotype (e.g., looking for homozygous variants in family trios). Unfortunately, we cannot directly do this in the SQL query, but the *gemini query* tool has an option called *–gt-filter* that allows one to specify filters to apply to the returned rows. The rules followed in the *–gt-filter* option follow Python syntax.

Tip: As you will see from the examples below, appropriate use of the –gt-filter option will allow you to compose queries that return variants meeting inheritance patterns that are relevant to the disease model of interest in your study.

As an example, let's only return rows where sample 1094PC0012 is heterozygous. In order to do this, we apply a filter to the *gt_types* columns for this individual:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene,
                       gts.1094PC0005, \
                       gts.1094PC0009, \
                       gts.1094PC0012, \
                       gts.1094PC0013 \
                from variants" \
                --gt-filter "gt_types.1094PC0012 == HET" \
                --header \
                test.snpEff.vcf.db

chrom   start   end     ref     alt     gene gts.1094PC0005     gts.1094PC0009  gts.
↪1094PC0012  gts.1094PC0013
```

```
chr1    30866   30869   CCT     C       FAM138A CCT/CCT CCT/CCT CCT/C   CCT/CCT
chr1    69510   69511   A       G       OR4F5   ./.     ./.     A/G     A/G
```

Now let's be a bit less restrictive and return variants where either sample 1094PC0012 is heterozygous or sample 1094PC0005 is homozygous for the reference allele:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene,
                       gts.1094PC0005, \
                       gts.1094PC0009, \
                       gts.1094PC0012, \
                       gts.1094PC0013 \
                from variants" \
                --gt-filter "gt_types.1094PC0012 == HET or \
                gt_types.1094PC0005 == HOM_REF" \
                --header \
                test.snpEff.vcf.db

chrom   start   end     ref     alt     gene gts.1094PC0005     gts.1094PC0009  gts.
↪1094PC0012  gts.1094PC0013
chr1    30859   30860   G       C       FAM138A G/G     G/G     G/G     G/G
chr1    30866   30869   CCT     C       FAM138A CCT/CCT CCT/CCT CCT/C   CCT/CCT
chr1    69427   69428   T       G       OR4F5   T/T     T/T     T/T     T/T
chr1    69510   69511   A       G       OR4F5   ./.     ./.     A/G     A/G
chr1    69760   69761   A       T       OR4F5   A/A     A/T     A/A     A/A
```

Eh, I changed my mind, let's restrict the above to those variants where sample 1094PC0012 must also be heterozygous:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene,
                       gts.1094PC0005, \
                       gts.1094PC0009, \
                       gts.1094PC0012, \
                       gts.1094PC0013 \
                from variants" \
                --gt-filter "(gt_types.1094PC0012 == HET or \
                gt_types.1094PC0005 == HOM_REF) \
                and \
                (gt_types.1094PC0013 == HET)" \
                --header \
                test.snpEff.vcf.db

 chrom  start   end     ref     alt     gene gts.1094PC0005     gts.1094PC0009  gts.
↪1094PC0012  gts.1094PC0013
```

`--gt-filter` Wildcard filtering on genotype columns.

Many times, we want to be able to apply the same rule to multiple samples without having to enter the rule over and over again for each sample. For example, let's imaging there are 100 samples in your study and you only want to report variants where every sample has an observed alignment depth of at least 20 reads. Traditionally, one would have enter each of the 100 samples from the command line as follows:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene from variants" \
                    --gt-filter "gt_depths.sample1 >= 20 and \
                                 gt_depths.sample2 >= 20 and \
                                 gt_depths.sample3 >= 20 and \
                                 ...
```

```
                                    gt_depths.sample100 >= 20" \
                        extended_ped.db
```

The basics.

Obviously, this is deeply painful. Thankfully, there an option allowing the use of "wildcards" to prevent this.

Note: The syntax of the wildcard `--gt-filters` is `(COLUMN).(SAMPLE_WILDCARD).(SAMPLE_WILDCARD_RULE).(RULE_ENFORCEMENT).`

For example, using wildcards, the above could be converted to:

```
$ gemini query -q "select chrom, start, end, ref, alt, gene from variants" \
                    --gt-filter "(gt_depths).(*).(>=20).(all)" \
                    extended_ped.db
```

Obviously, this makes things much simpler.

The `all` operator

One can also apply wildcards that select samples based on the values in specific columns in the `samples` table. For example, let's imagine we wanted to require that variants are returned only in cases where *ALL* the affected individuals in the study (i.e., the `phenotype` column in the `samples` table is equal to `2`) have non-reference genotypes. We could do the following:

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, (gts).(phenotype==2) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==2).(!=HOM_REF).(all)" \
               extended_ped.db
chrom start end ref alt gene  gts.M10478  gts.M10500
chr10 1142207 1142208 T C WDR37 C/C C/C
chr10 48003991  48003992  C T ASAH2C  C/T C/T
chr10 52004314  52004315  T C ASAH2 ./. C/C
chr10 52497528  52497529  G C ASAH2B  C/C C/C
chr10 135210790 135210791 T C MTG1.1  C/C C/C
chr10 135336655 135336656 G A SPRN  A/A ./.
chr10 135369531 135369532 T C SYCE1 T/C T/C
```

Or perhaps we wanted to be more restrictive. We could also enforce that the affected individuals also had at least 20 aligned reads at such variant sites:

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, \
                         (gts).(phenotype==2), (gt_depths).(phenotype==2) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==2).(!=HOM_REF).(all) \
                            and \
                            (gt_depths).(phenotype==2).(>=20).(all)" \
               extended_ped.db
chrom start end ref alt gene  gts.M10478  gts.M10500  gt_depths.M10478  gt_depths.
↪M10500
chr10 1142207 1142208 T C WDR37 C/C C/C 29  24
```

```
chr10 48003991  48003992  C T ASAH2C  C/T C/T 38  44
chr10 135369531 135369532 T C SYCE1 T/C T/C 32  21
```

The `any` operator

The examples provided thus far have deomnstrated how to enforce that *ALL* of the samples meeting specific criteria meet the same genotype column restrictions. However, clearly there are cases where one would want to be less restrictive. In order to accomondate such queries, there are three other enforcement operators allowed: `any`, `none`, and `count`.

For example, perhaps we want to relax the above query a bit and only require that at least one (i.e., `any`) of the affected samples has depth > 20:

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, \
                         (gts).(phenotype==2), (gt_depths).(phenotype==2) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==2).(!=HOM_REF).(all) \
                            and \
                            (gt_depths).(phenotype==2).(>=20).(any)" \
               extended_ped.db
```

The `none` operator

Or we enforce that `none` of the affected samples have depth less than 10:

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, \
                         (gts).(phenotype==2), (gt_depths).(phenotype==2) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==2).(!=HOM_REF).(all) \
                            and \
                            (gt_depths).(phenotype==2).(<10).(none)" \
               extended_ped.db
```

The `count` operator

Finally, we could enforce that at most 2 of all the samples in the study have depths < 10:

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, \
                         (gts).(phenotype==2), (gt_depths).(phenotype==2) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==2).(!=HOM_REF).(all) \
                            and \
                            (gt_depths).(phenotype==2).(<10).(count <= 2)" \
               extended_ped.db
```

Note: The `count` operator allows the following comparisons: >, >=, <, <=, ==, <>, and !=.

Example scenario.

One usage of the wildcard functionality is screening for variants that are present in affected individuals yet absent from unaffected individuals (this is obviously an idealized scenario).

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, (gts).(*) \
                   FROM variants" \
               --gt-filter "(gt_types).(phenotype==1).(==HOM_REF).(all) \
                            and \
                            (gt_depths).(phenotype==2).(==HOM_REF).(none)" \
               extended_ped.db
```

More complexity.

One can also combine wildcard filters with "basic" genotype filters to create more complex logic.

```
$ gemini query --header \
               -q "SELECT chrom, start, end, ref, alt, gene, (gts).(*) \
                   FROM variants" \
               --gt-filter "((gt_types).(phenotype==1).(==HOM_REF).(all) \
                             and \
                            (gt_depths).(phenotype==2).(==HOM_REF).(none)) \
                           or gt.NA12878 == HET" \
               extended_ped.db
```

Other details.

The system is fairly flexible in that it allows one to wildcard-select samples based on custom columns that have been added to the `samples` table based upon a custom PED file. For example, let's imaging our custom PED file had an extra column defining the hair color of each sample. We coukd use that to restrict interesting variants to those where samples with blue hair were heterozygous:

```
$ gemini query -q "SELECT chrom, start, end, ref, alt, gene FROM variants" \
               --gt-filter "(gt_types).(hair_color='blue').(==HET).(all)" \
               extended_ped.db
```

Or possibly, you want to stratify based on sub-population:

```
$ gemini query -q "SELECT chrom, start, end, ref, alt, gene FROM variants" \
               --gt-filter "(gt_types).(population='CEU').(==HET).(all) \
                            and \
                           (gt_types).(population='YRI').(==HOM_ALT).(any)" \
               extended_ped.db
```

One can also base the wildcard on multiple criteria (in this case, brown hair and affected):

```
$ gemini query -q "SELECT chrom, start, end, ref, alt, gene FROM variants" \
               --gt-filter "(gt_types).(hair_color=='brown' and phenotype==2).(!=
↪HET).(all)" \
               extended_ped.db
```

There is a special case when we want to select samples based on their genotypes. For example, to show only sites where HET samples have a genotype quality above 20.

```
$ gemini query -q "SELECT chrom, start, end, ref, alt, gene FROM variants \
                where gene == 'SCNN1D' limit 5" \
                 --gt-filter "(gt_quals).(=HET).(<20).(all)" test/test.query.db
```

In this case, the 2nd value must be one of *=HET*, *=HOM_REF* or *=HOM_ALT*

Lastly, wildcards can, of course, be combined with non-wildcard criteria:

```
$ gemini query -q "SELECT chrom, start, end, gene FROM variants" \
                --gt-filter "(gt_types).(hair_color=='brown' and phenotype==2).(!=␣
↪HET).(all) \
                             and \
                            gt_types.M128215 == HOM_REF" \
                extended_ped.db
```

Hopefully this gives you a sense of what you can do with the "wildcard" genotype filter functionality.

`--show-samples` Finding out which samples have a variant

While exploring your data you might hit on a set of interesting variants and want to know which of your samples have that variant in them. You can display the samples containing a variant with the –show-sample-variants flag:

```
$ gemini query --header --show-samples -q "select chrom, start, end, ref, alt \
                                from variants where is_lof=1 limit 5" test.query.db

chrom   start   end     ref     alt     variant_samples het_samples     hom_alt_
↪samples
chr1    874815  874816  C       CT      1478PC0006B,1478PC0007B,1478PC0010,
↪1478PC0013B,1478PC0022B,1478PC0023B,1478PC0025,1719PC0007,1719PC0009,1719PC0010,
↪1719PC0022   1478PC0006B,1478PC0007B,1478PC0010,1478PC0013B,1478PC0022B,1478PC0023B,
↪1719PC0007,1719PC0009,1719PC0010 1478PC0025,1719PC0022
chr1    1140811 1140813 TC      T       1478PC0011      1478PC0011
chr1    1219381 1219382 C       G       1719PC0012      1719PC0012
chr1    1221487 1221490 CAA     C       1478PC0004      1478PC0004
```

variant_samples is a list of all of the samples with a variant, het_samples is the subset of those heterozygous for the variant and hom_alt_samples is the subset homozygous for the variant.

`--show-samples --format sampledetail` Provide a flattened view of samples

If you'd like to be able to export variants plus associated sample metadata into a downstream tool like R or pandas for additional exploration, adding the `--format sampledetail` command flattens all found samples and attached metadata information:

```
$ gemini query --header --format sampledetail --show-samples \
   -q "select chrom, start, ref from variants where is_lof=1 limit 1" test.query.db

chrom       start   ref     family_id       name    paternal_id     maternal_id     ␣
↪sex    phenotype
chr1        30547   T       0       1478PC0016      0       0       -9      -9
chr1        30547   T       0       1719PC0007      0       0       -9      -9
chr1        30547   T       0       1719PC0009      0       0       -9      -9" > exp
```

The denormalized results contain a row for each sample associated with a variant, along with information from the sample table. This provides a way to join sample information with a variant query.

`--show-families` Finding out which families have a variant

This works exactly like `--show-samples` except lists all of the families with a variant instead of the individual samples.

`--region` Restrict a query to a specified region

If you are only interested in a specific region, you can restrict queries to that region using the `--region` tool.

```
$ gemini query --region chr1:30859-30900 -q "select chrom, start, end, ref, alt \
             from variants"  test1.snpeff.db
chr1 30859   30860   G       C
```

`--sample-filter` Restrict a query to specified samples

The `--sample-filter` option allows you to select samples that a variant must be in by doing a SQL query on the samples table. For example if you wanted to show the set of variants that appear in all samples with a phenotype status of 2, you could do that query with:

```
$ gemini query --sample-filter "phenotype=2" -q "select gts, gt_types from variants"␣
↪test.family.db
T/T,T/T,T/C,T/T,T/T,T/T,T/T,T/T,C/C  0,0,1,0,0,0,0,0,3       1_kid,3_kid     1_kid  ␣
↪3_kid
T/T,T/T,T/C,T/T,T/T,T/C,T/T,T/T,T/C  0,0,1,0,0,1,0,0,1       1_kid,2_kid,3_kid      ␣
↪1_kid,2_kid,3_kid
T/T,T/T,T/T,T/T,T/T,T/T,T/T,T/T,T/C  0,0,0,0,0,0,0,0,1       3_kid   3_kid
```

By default –sample-filter will show the variant if at least one sample contains the variant. You can change this behavior by using the `--in` option along with `--sample-filter`. `--in all` will return a variant if all samples matching the query have the variant. `in none` will return a variant if the variant does not appear in any of the matching samples. `--in only` will return a variant if the variant is only in the matching samples and not in any of the non-matching samples. `--in only all` will show all of the variant which are in all of the matching samples and not in any of the non-matching samples.

The `--family-wise` flag applies the `--sample-filter` and `--in` behavior on a family-wise basis. For example to show all variants that are only in samples with a phenotype status of 2 in at least one family:

```
$ gemini query --family-wise --in only all --sample-filter "phenotype=2" -q "select␣
↪gts, gt_types from variants" test.family.db
T/T,T/T,T/C,T/T,T/T,T/T,T/T,T/T,C/C  0,0,1,0,0,0,0,0,3       1_kid,3_kid     1_kid  ␣
↪3_kid
T/T,T/T,T/C,T/T,T/T,T/C,T/T,T/T,T/C  0,0,1,0,0,1,0,0,1       1_kid,2_kid,3_kid      ␣
↪1_kid,2_kid,3_kid
T/T,T/T,T/T,T/T,T/T,T/T,T/T,T/T,T/C  0,0,0,0,0,0,0,0,1       3_kid   3_kid
```

You can also specify that a variant passes this filter in multiple families with the `--min-kindreds` option. So if you want to do the same query above, but restrict it such that at least three families have to pass the filter:

```
$ gemini query --min-kindreds 3 --family-wise --in only all --sample-filter
↪"phenotype=2" -q "select gts, gt_types from variants" test.family.db
T/T,T/T,T/C,T/T,T/T,T/C,T/T,T/T,T/C  0,0,1,0,0,1,0,0,1       1_kid,2_kid,3_kid      ␣
↪1_kid,2_kid,3_kid
```

If the PED file you loaded has extra fields in it, those will also work with the `--sample-filter` option. For example if you had a `hair_color` extended field, you could query on that as well as phenotype:

```
$ gemini query  --in only all --sample-filter "phenotype=1 and hair_color='blue'" -q
↪"select gts, gt_types from variants" extended_ped.db
G/G,G/G,G/G,G/A      0,0,0,1 M128215 M128215
```

`--sample-delim` Changing the sample list delimiter

One can modify the default comma delimiter used by the `--show-samples` option through the use of the `--sample-delim` option. For example, to use a semi-colon instead of a comma, one would do the following:

```
  $ gemini query --header --show-samples --sample-delim ";" \
                 -q "select chrom, start, end, ref, alt \
                     from variants where is_lof=1 limit 5" test.query.db

chrom start end ref alt variant_samples het_samples hom_alt_samples
chr1  874815  874816  C CT  1478PC0006B;1478PC0007B;1478PC0010,1478PC0013B;
↪1478PC0022B;1478PC0023B;1478PC0025;1719PC0007;1719PC0009;1719PC0010;1719PC0022␣
↪1478PC0006B;1478PC0007B;1478PC0010;1478PC0013B;1478PC0022B;1478PC0023B;1719PC0007;
↪1719PC0009;1719PC0010 1478PC0025;1719PC0022
chr1  1140811 1140813 TC  T 1478PC0011  1478PC0011
chr1  1219381 1219382 C G 1719PC0012  1719PC0012
chr1  1221487 1221490 CAA C 1478PC0004  1478PC0004
```

`--format` Reporting query output in an alternate format.

The results of GEMINI queries can automatically be formatted for use with other programs using the –format command. Supported alternative formats are JSON and TPED (Transposed PED) format.

Reporting query output in JSON format may enable HTML/Javascript apps to query GEMINI and retrieve the output in a format that is amenable to web development protocols.

Here is a basic query:

```
$ gemini query -q "select chrom, start, end from variants" my.db | head
chr1  10067 10069
chr1  10230 10231
chr1  12782 12783
chr1  13109 13110
chr1  13115 13116
chr1  13117 13118
chr1  13272 13273
chr1  13301 13302
chr1  13416 13417
chr1  13417 13418
```

To report in JSON format, use the `--format json` option. For example:

```
$ gemini query --format json -q "select chrom, start, end from variants" my.db | head
{"chrom": "chr1", "start": 10067, "end": 10069}
{"chrom": "chr1", "start": 10230, "end": 10231}
{"chrom": "chr1", "start": 12782, "end": 12783}
{"chrom": "chr1", "start": 13109, "end": 13110}
{"chrom": "chr1", "start": 13115, "end": 13116}
{"chrom": "chr1", "start": 13117, "end": 13118}
{"chrom": "chr1", "start": 13272, "end": 13273}
{"chrom": "chr1", "start": 13301, "end": 13302}
```

```
{"chrom": "chr1", "start": 13416, "end": 13417}
{"chrom": "chr1", "start": 13417, "end": 13418}
```

If you would to use tools such as PLINK that use the PED format, you can dump out a set of variants matching any query in TPED (Transposed PED) format by adding the `--tped` flag to your query:

```
$ gemini query --format tped -q "select * from variants where chrom=10" test4.snpeff.
↪db
10 rs10794716 0 1142207 C/C C/C C/C C/C
10 rs142685947 0 48003991 T/T C/T C/T C/C
10 rs2842123 0 52004314 ./. ./. C/C C/C
10 rs4935178 0 52497528 ./. C/C C/C ./.
16 rs201947120 0 72057434 C/T C/C C/C C/C
10 rs73373169 0 126678091 G/G G/G G/G G/A
10 rs2265637 0 135210790 T/T C/C C/C T/T
10 rs6537611 0 135336655 ./. A/A ./. A/A
10 rs3747881 0 135369531 T/T T/C T/C T/T
```

You can pass –header to get a header to see which samples have which variant. To use the TPED format you also need to generate a corresponing TFAM file from your data as well, which you can get from the GEMINI dump tool:

```
$ gemini dump  --tfam test4.snpeff.db > obs
None    M10475  None    None    None    None
None    M10478  None    None    None    None
None    M10500  None    None    None    None
None    M128215 None    None    None    None
```

`--carrier-summary-by-phenotype` Summarize carrier status

For prioritizing variants sometimes it is useful to have summary counts of the carrier status for all samples with a variant stratified across a phenotype. `--carrier-summary-by-phenotype` takes a column in the samples table that you want to summarize the carrier status of and adds a set of counts of carrier/non-carrier status for each phenotype in the given column. For example, to get a summary of how a set of variants segregate with affected status:

```
$ gemini query --show-samples --carrier-summary-by-phenotype affected --header -q
↪"select chrom, start, ref, alt, gt_types from variants" extended_ped_test.db
chrom   start   ref     alt     gt_types        variant_samples het_samples     hom_
↪alt_samples unaffected_carrier      affected_carrier        unaffected_noncarrier
↪affected_noncarrier     unknown
chr10   1142207 T       C       3,3,3,3 M10475,M10478,M10500,M128215
↪M10475,M10478,M10500,M128215    2       2       0       0       0
chr10   48003991        C       T       3,1,1,0 M10475,M10478,M10500    M10478,M10500
↪  M10475  1       2       1       0       0
chr10   52004314        T       C       2,2,3,3 M10500,M128215          M10500,
↪M128215  1       1       0       0       2
chr10   52497528        G       C       2,3,3,2 M10478,M10500           M10478,M10500
↪  0       2       0       0       2
chr16   72057434        C       T       1,0,0,0 M10475  M10475          1       0
↪  1       2       0
chr10   126678091       G       A       0,0,0,1 M128215 M128215         1       0
↪  1       2       0
chr10   135210790       T       C       0,3,3,0 M10478,M10500           M10478,M10500
↪  0       2       2       0       0
chr10   135336655       G       A       2,3,2,3 M10478,M128215          M10478,
↪M128215  1       1       0       0       2
chr10   135369531       T       C       0,1,1,0 M10478,M10500   M10478,M10500
↪  0       2       2       0       0
```

Or if you have another phenotypic feature you are interested in summarizing, like hair color:

```
$ gemini query --show-samples --carrier-summary-by-phenotype hair_color --header -q
↪"select chrom, start, ref, alt, gt_types from variants" extended_ped.db
chrom   start   ref     alt     gt_types        variant_samples het_samples     hom_
↪alt_samples blue_carrier    brown_carrier   purple_carrier  blue_noncarrier brown_
↪noncarrier        purple_noncarrier       unknown
chr10   1142207 T       C       3,3,3,3 M10475,M10478,M10500,M128215
↪M10475,M10478,M10500,M128215    1       2       1       0       0       0       0
chr10   48003991        C       T       3,1,1,0 M10475,M10478,M10500    M10478,M10500
↪  M10475  0       2       1       1       0       0       0
chr10   52004314        T       C       2,2,3,3 M10500,M128215          M10500,
↪M128215  1       0       1       0       0       0       2
chr10   52497528        G       C       2,3,3,2 M10478,M10500           M10478,M10500
↪  0       1       1       0       0       0       2
chr16   72057434        C       T       1,0,0,0 M10475  M10475          0       1
↪  0       1       1       1       0
chr10   126678091       G       A       0,0,0,1 M128215 M128215         1       0
↪  0       0       2       1       0
chr10   135210790       T       C       0,3,3,0 M10478,M10500           M10478,M10500
↪  0       1       1       1       1       0       0
chr10   135336655       G       A       2,3,2,3 M10478,M128215          M10478,
↪M128215  1       1       0       0       0       0       2
chr10   135369531       T       C       0,1,1,0 M10478,M10500   M10478,M10500
↪  0       1       1       1       1       0       0
```

Querying the gene tables

The gene tables viz. `gene_detailed table` and the `gene_summary table` have been built on version 73 of the ensembl genes. The column specifications are available at *The GEMINI database schema*. These tables contain gene specific information e.g. gene synonyms, RVIS percentile scores(Petrovski et.al 2013), strand specifications, cancer gene census etc. While the former is more detailed, the later lacks transcript wise information and summarizes some aspects of the former. For e.g. while the gene_detailed table lists all transcripts of a gene with their start and end co-ordinates, the gene_summary table reports only the minimum start and maximum end co-ordinates of the gene transcripts. The `chrom`, `gene` and the `transcript` columns of the gene tables may be used to join on the variants and the variant_impacts tables.

Using the `gene_detailed` & `gene_summary` tables

Query the gene_detailed table with a join on variants table:

E.g. Get additional transcript info for the most severe impact transcript e.g. transcript status, transcript start,end and the protein length

```
$ gemini query --header -q "select v.variant_id, v.gene, \
                    v.impact, g.transcript_status, g.transcript, \
                              g.transcript_start, g.transcript_end, g.protein_length,
↪ \
                              from variants v, gene_detailed g \

                              WHERE v.chrom = g.chrom AND \
                                          v.gene = g.gene AND \
```

```
                                        v.transcript = g.transcript AND \
                                        v.impact_severity='HIGH'" test.query.db

   variant_id      gene    impact  transcript_status       transcript      ␣
↪transcript_start       transcript_end  protein_length
   46      SAMD11  frame_shift     KNOWN   ENST00000342066 861118  879955  681
   578     TNFRSF18        frame_shift     PUTATIVE        ENST00000486728 1139224␣
↪1141060 169
   733     SCNN1D  stop_gain       NOVEL   ENST00000470022 1217305 1221548 138
```

Query the gene_detailed table with a join on the variant_impacts table:

E.g. Get the transcript status for all transcripts of the SCNN1D gene where impact severity is not 'LOW'.

```
$ gemini query --header -q "select v.gene, g.transcript_status, g.transcript, \
               v.impact from variant_impacts v, gene_detailed g \

                           WHERE v.transcript = g.transcript AND \
                    v.gene = g.gene AND \
                                        v.gene = 'SCNN1D' \
                        v.impact_severity!='LOW'" test.query.db

   gene    transcript_status       transcript      impact
   SCNN1D  NOVEL   ENST00000470022 non_syn_coding
   SCNN1D  NOVEL   ENST00000470022 frame_shift
   SCNN1D  KNOWN   ENST00000325425 frame_shift
   SCNN1D  KNOWN   ENST00000379116 non_syn_coding
   SCNN1D  KNOWN   ENST00000338555 non_syn_coding
   SCNN1D  KNOWN   ENST00000400928 non_syn_coding
```

Query the gene_summary table with a join on the variants table:

E.g. Get the synonym/alternate names, RVIS percentile scores and the min-max start-end of transcripts for genes that have a severely affected transcript of a 'HIGH' order.

```
$ gemini query --header -q "select v.chrom, v.gene, g.transcript_min_start, \
                   g.transcript_max_end, g.synonym, g.rvis_pct, v.impact from \
                               variants v, gene_summary g \

                               WHERE v.chrom = g.chrom AND \
                     v.gene = g.gene AND \
                     v.impact_severity='HIGH'" test.query.db

   chrom   gene    transcript_min_start    transcript_max_end      synonym rvis_pct ␣
↪       impact
   chr1    SAMD11  860260  879955  MGC45873        None    frame_shift
   chr1    TNFRSF18        1138888 1142071 AITR,CD357,GITR None    frame_shift
   chr1    SCNN1D  1215816 1227409 ENaCdelta,dNaCh 96.77990092     stop_gain
```

Query the gene_summary table with a join on the variant_impacts table:

E.g. Get all variants of a gene, the affected transcripts and impacts, where a mammalian phenotype ID is available for the `mouse phenotype`.

```
$ gemini query --header -q "select v.variant_id, v.chrom, v.gene, i.impact, \
                  i.transcript, g.mam_phenotype_id from variants v, \
                              variant_impacts i, gene_summary g \

                              WHERE v.variant_id=i.variant_id \
                              i.gene=g.gene AND \
                              v.chrom=g.chrom AND \
                              g.mam_phenotype_id !='None'" test.query.db

    variant_id      chrom   gene    impact  transcript      mam_phenotype_id
    334     chr1    TNFRSF18        non_syn_coding  ENST00000328596 MP:0005397,
↪MP:0005384,MP:0005387
    378     chr1    TNFRSF18        frame_shift     ENST00000486728 MP:0005397,
↪MP:0005384,MP:0005387
    483     chr1    AGRN    synonymous_coding       ENST00000379370 MP:0005378,
↪MP:0005386,MP:0005388,MP:0005367,MP:0005369,MP:0005371,MP:0003631,MP:0002873,
↪MP:0010768
    484     chr1    AGRN    exon    ENST00000461111 MP:0005378,MP:0005386,MP:0005388,
↪MP:0005367,MP:0005369,MP:0005371,MP:0003631,MP:0002873,MP:0010768
    478     chr1    AGRN    intron  ENST00000461111 MP:0005378,MP:0005386,MP:0005388,
↪MP:0005367,MP:0005369,MP:0005371,MP:0003631,MP:0002873,MP:0010768
    479     chr1    AGRN    downstream      ENST00000492947 MP:0005378,MP:0005386,
↪MP:0005388,MP:0005367,MP:0005369,MP:0005371,MP:0003631,MP:0002873,MP:0010768
```

Restrict analysis to transcripts with a valid `CCDS_ID`

Since the current available transcript sets are more than one (e.g. RefSeq, ENSEMBL and UCSC) we support information (e.g pathways tool) for the ENSEMBL transcripts but provide a mapping of these transcripts to the consensus set agreed upon by all the above three mentioned groups viz. `transcripts having a valid CCDS_ID`. Here we show, how we can return variants and their impacts for only these restricted set of transcripts using the gene_detailed table.

```
$ gemini query --header -q "select i.var_id, i.gene, i.impact, i.transcript, \
                g.transcript_status, ccds_id, g.rvis_pct from \
                              variant_impacts i, gene_detailed g where \
                              i.transcript=g.transcript and i.gene=g.gene and\
                              impact_severity='HIGH' and g.ccds_id!='None'" test.
↪query.db

variant_id      gene    impact  transcript      transcript_status       ccds_id rvis_
↪pct
2051    SAMD11  frame_shift     ENST00000342066 KNOWN   CCDS2   None
3639    CCNL2   splice_acceptor ENST00000408918 KNOWN   CCDS30558       53.98089172
3639    CCNL2   splice_acceptor ENST00000400809 KNOWN   CCDS30557       53.98089172
13221   SMIM1   frame_shift     ENST00000444870 NOVEL   CCDS57966       None
21881   NPHP4   splice_acceptor ENST00000378156 KNOWN   CCDS44052       81.78815758
```

What if I don't see my `gene` in the database?

Most genes are popular by their common names while the representation of gene names in the GEMINI database is mostly HGNC. For e.g `ARTEMIS` would be `DCLRE1C` in the GEMINI database. As such one may miss out on variants if looking for specific genes by their common names. While, joining the main tables with the gene tables for synonym information would be useful (as shown in the previous examples), the gene tables may also serve as a quick look up for alternate names of a gene, which could then be looked up in the database.

```
$ gemini query -q "select synonym from gene_summary where \
                   gene='ARTEMIS'" test.query.db
A-SCID,FLJ11360,SNM1C,DCLRE1C,SCIDA

or,

$ gemini query -q "select gene from gene_summary where synonym \
                   like '%ARTEMIS%' and \
                   is_HGNC='1'" test.query.db
DCLRE1C

#looking up for DCLRE1C in the database
$ gemini query -q "select variant_id, chrom, start, end, impact \
                   from variants where \
                   gene='DCLRE1C'" test.query.db
```

Built-in analysis tools

As of version 0.14, the tools have a standardized output that is different from previous versions. Requested *–columns* will come first followed by a standard set of columns:

- variant_id - unique id from the databse
- family_id - family id for this row
- family_members - which family members were tested
- family_genotypes - genotypes of this family
- samples - samples contributing to this row appearing in the results
- family_count - number of families with this effect

Other tools such as *mendel_errors* additional columns at the end.

When queries are limited to variants in genes, the output will be in *gene*, *chrom* order as opposed to the usual *gene*, *position* order.

Note: As of version 0.16.0, the inheritance tools (autosomal_dominant, autosomal_recessive, comp_het, mendel_errors, de_novo) are now more strict by default.

A –lenient flag allows, e.g. allows some samples to be of unknown phenotype or to not have both parents of known phenotype.

The –allow-unaffected flag will result in reporting variants where unaffected samples share the same variants. The default will only report variants that are unique to affected samples.

The design decisions for this change are described here: https://github.com/arq5x/gemini/issues/388 a visual representation is here: https://github.com/arq5x/gemini/blob/master/inheritance.ipynb

---

Note: Candidate variants reported by the built-in inheritance model tools will appear in order by chromosome, then alphabetically by gene. In other words, they will not be in strict positional order for each chromosome. This is in an effort to group all candidate variants by gene since the gene is typically the atomic unit of interest.

Additionally, all current built-in tools (autosomal_dominant, comp_hets, and autosomal_recessive) only analyze autosomal (non sex chromosome) variants. Analysis of X- and Y-linked phenotypes must be done manually with *–gt-filter*

`common_args`: common arguments

The inheritance tools share a common set of arguments. We will describe them here and refer to them in the corresponding sections:

`--columns`

This flag is followed by a comma-delimited list of columns the user is requestin in the output.

`--min-kindreds 1`

This is the number of families required to have a variant in the same gene in order for it to be reported. For example, we may only be interested in candidates where at least 4 families have a variant in that gene.

`--families`

By default, candidate variants are reported for all families in the database. One can restrict the analysis to variants in specific familes with the `--families` option. Families should be provided as a comma-separated list

`--filter`

By default, each tool will report all variants regardless of their putative functional impact. In order to apply additional constraints on the variants returned, one can use the `--filter` option. Using SQL syntax, conditions applied with the `--filter` options become WHERE clauses in the query issued to the GEMINI database.

`-d [0]` (depth)

Filter variants that do not have at least this depth for all members in a a family. Default is 0.

`--min-gq [0]`

Filter variants that do not have at least this genotype quality for each sample in a family. Default is 0. Higher values are more stringent.

`--allow-unaffected`

By default, candidates that also appear in unaffected samples are not reported if this flag is specified, such variants will be reported.

`--lenient`

Loosen the restrictions on family structure. This will allow, for example, finding compound_hets in unaffected samples.

`--gt-pl-max`

In order to eliminate less confident genotypes, it is possible to enforce a maximum PL value for each sample. On this scale, lower values indicate more confidence that the called genotype is correct. 10 is a reasonable value. This is applied per-family such that all members of a family must meet this level in order to by reported in the final results.

`comp_hets`: Identifying potential compound heterozygotes

Many autosomal recessive disorders are caused by compound heterozygotes. Unlike canonical recessive sites where the same recessive allele is inherited from both parents at the _same_ site in the gene, compound heterozygotes occur when the individual's phenotype is caused by two heterozygous recessive alleles at _different_ sites in a particular gene.

We are looking for two (typically loss-of-function (LoF)) heterozygous variants impacting the same gene at different loci. The complicating factor is that this is _recessive_ and as such, we must also require that the consequential alleles at each heterozygous site were inherited on different chromosomes (one from each parent). Where possible, *comp_hets* will phase by transmission. Once this has been done, the *comp_hets* tool will provide a report of candidate compound heterozygotes for each sample/gene.

Non-exonic/non-coding analyses: *comp_hets* excludes intronic/non-coding variants for which *impact_severity == 'LOW' AND is_exonic == FALSE*. Therefore, *comp_hets* will not retrieve most pairs of variants that are downstream or upstream of a gene or are intronic unless otherwise annotated with medium or high *impact_severity*.

Note: As of version 0.16.0 the `comp_het` tool will perform family-based phasing by default in order to provide better candidates even in the absence of unphased genotypes. Any candidate that could be one element of a comp_het will also be phaseable as long as the parents and their genotypes are known.

As of version 0.16.1, the –ignore-phasing option is removed and there is no –lenient option.

In 0.16.2, a –pattern-only flag was added to find compound hets by inheritance pattern without regard to affection status. A priority code was also added where variants with priority 1 are much more informative. See docs below for further information.

Genotype Requirements

- All affected individuals must be heterozygous at both sites.
- No unaffected can be homozygous alterate at either site.
- Neither parent of an affected sample can be homozygous reference at both sites.
- If any unphased-unaffected is het at both sites, the site will be give lower priority
- No phased-unaffected can be heterozygous at both sites.
    1. *–allow-unaffected* keeps sites where a phased unaffected shares the het-pair
    2. unphased, unaffected that share the het pair are counted and reported for each candidate pair.
- Remove candidates where an affected from the same family does NOT share the same het pair.
- Sites are automatically phased by transmission when parents are present in order to remove false positive candidates.
    1. If data from one or both parents are unavailable and the child's data was not phased prior to loading into GEMINI, all comp_het variant pairs will automatically be given at most priority == 2. If there's only a single parent and both the parent and the affected are HET at both sites, the candidate will have priority 3.

2. *–max-priority x* can be used to set the maximum allowed priority level at which candidate pairs are included in the output.

we prioritize with these rules:

mom	dad	kid	phaseable	priority	notes
R-H	H-R	H-H	both	1	both sites phaseable and alts on opposite chroms
n/a	n/a	H-H	NO	2	singleton (unphaseable) HETs have priority 2.
R-H	H-H	H-H	one	3	should be a rare occurrence
H-H	H-H	H-H	NO	3	should be a rare occurrence
H-H	UNK	H-H	NO	3	missing parent and all hets.
A-R	H-H	H-H	both	NA	exclude hom-alts from un-affecteds
R-R	H-H	H-H	both	NA	phaseable, but alts are on the same chroms.

Note: candidates of priority == 3 are very unlikely (< 1%) to be causal for a rare Mendelian condition (see: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3734130/); we report them for completeness, but strongly recommend using priority 1 and 2 only. Priority 2 is useful when there are multiple families, some of which consist of only a single sequenced, affected sample.

Pattern Only

To find compound heterozygotes by inheritance pattern only, without regard to affections, the following rules are used (with –pattern-only):

- Kid must be HET at both sites.
- Kid must have alts on different chromosomes.
- Neither parent can be HOM_ALT at either site.
- If either parent is phased at both sites and matches the kid, it's excluded.
- If either parent is HET at both sites, priority is reduced.
- When the above criteria are met, and both parents and kid are phased or parents are HET at different sites, the priority is 1.
- If both parents are not phased, the priority is 2.
- For every parent that's a het at both sites, the priority is incremented by 1.
- The priority in a family is the minimum found among all kids.

Note: Each pair of consecutive lines in the output represent the two variants for a compound heterozygote in a give sample. The third column, *comp_het_id*, tracks the distinct compound heterozygote variant pairs.

Example usage with a subset of columns:

```
$ gemini comp_hets my.db --columns "chrom, start, end" test.comp_het_default.2.db
chrom   start   end gene    alt variant_id  family_id   family_members  family_
↪genotypes    samples family_count    comp_het_id
chr1    17362   17366   WASH7P  T   1   3   dad_3(dad;unaffected),mom_3(mom;
↪unaffected),child_3(child;affected) TTCT|T,TTCT|TTCT,TTCT|T child_3 2   1
chr1    17729   17730   WASH7P  A   2   3   dad_3(dad;unaffected),mom_3(mom;
↪unaffected),child_3(child;affected) C|A,C|A,A|C child_3 2   1
```

```
chr1    17362   17366   WASH7P  T   1   4   dad_4(dad;unaffected),mom_4(mom;
↪unaffected),child_4(child;affected) TTCT|T,TTCT|TTCT,TTCT|T child_4 2   1
chr1    17729   17730   WASH7P  A   2   4   dad_4(dad;unaffected),mom_4(mom;
↪unaffected),child_4(child;affected) C|A,C|A,A|C child_4 2   1
```

This indicates that samples child_3 and child_4 have a candidate compound heterozygotes in WASH.

the following command would further restrict candidate genes to those genes with a compound heterozygote in at least two families:

```
$ gemini comp_hets -d 50 \
      --columns "chrom, start, end, ref, alt" \
      --filter "impact_severity = 'HIGH'" \
      --allow-unaffected \
      --min-kindreds 2 \
      my.db
```

Now, this does not require that the family members are necessarily restricted to solely those that are affected. To impose this restriction, we remove the `--allow-unaffected` flag

```
$ gemini comp_hets -d 50 \
      --columns "chrom, start, end, ref, alt" \
      --filter "impact_severity = 'HIGH'" \
      --min-kindreds 2 \
      my.db
```

We may also specify the families of interest:

```
$ gemini comp_hets --families 1 my.db
$ gemini comp_hets --families 1,7 my.db
```

gene-where

The default selection of genes is by the clause: *"is_exonic = 1 or impact_severity != 'LOW"'* This can be specified to limit to a different subset, e.g. *"gene != ''*

`mendelian_error`: Identify non-mendelian transmission.

Note: This tool requires that you identify familial relationships via a PED file when loading your VCF into gemini via:

```
gemini load -v my.vcf -p my.ped my.db
```

We can query for mendelian errors in trios including:

- loss of heterozygosity
- implausible de-novo mutations
- de-novo mutations
- uniparental disomy

Genotype Requirements

- (LOH) kind and one parent are opposite homozygotes; other parent is HET
- (uniparental disomy) parents are opposite homozygotes; kid is homozygote;
- (plausible de novo) kid is het. parents are same homozygotes
- (implausible de novo) kid is homozygoes. parents are same homozygotes and opposite to kid.

If allow *–only-affected* is used, then the tools will only consider samples that have parents **and** are affected. The default is to consider any sample with parents.

This tool will report the probability of a mendelian error in the final column that is derived from the genotype likelihoods if they are available.

Example:

```
$ gemini mendel_errors --columns "chrom,start,end" test.mendel.db --gt-pl-max 1
chrom       start   end     variant_id      family_id       family_members  family_
↪genotypes          samples family_count    violation       violation_prob
chr1        10670   10671   1       CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        G/G,G/G,G/C     NA12877 1       plausible de␣
↪novo       0.962
chr1        28493   28494   2       CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        T/C,T/T,C/C     NA12877 1       loss of␣
↪heterozygosity  0.660
chr1        28627   28628   3       CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        C/C,C/C,C/T     NA12877 1       plausible de␣
↪novo       0.989
chr1        267558  267560  5       CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        C/C,C/C,CT/C    NA12877 1       plausible de␣
↪novo       0.896
chr1        537969  537970  7       CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        C/C,C/C,C/T     NA12877 1       plausible de␣
↪novo       0.928
chr1        547518  547519  11      CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        G/G,G/G,G/T     NA12877 1       plausible de␣
↪novo       1.000
chr1        589081  589086  14      CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        G/G,GAGAA/GAGAA,G/G     NA12877 1       ␣
↪uniparental disomy      0.940
chr1        749688  749689  16      CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        T/T,T/T,G/G     NA12877 1       implausible␣
↪de novo     0.959
chr1        788944  788945  17      CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        C/C,G/G,G/G     NA12877 1       uniparental␣
↪disomy      0.914
chr1        1004248 1004249 22      CEPH1463        NA12889(dad;unknown),NA12890(mom;
↪unknown),NA12877(child;unknown)        G/G,G/G,G/C     NA12877 1       plausible de␣
↪novo       1.000
```

Where, here, we have required the called genotype to have at most a PL of 1 (lower is more confident). Note that the "violation" column indicates the type of mendelian error and the final column can be used for further filtering, with higher numbers indicating a greater probability of mendelian error. We have found > 0.99 to be a reasonable cutoff.

Arguments are similar to the other tools:

```
positional arguments:
  db                    The name of the database to be queried.
```

```
optional arguments:
  -h, --help            show this help message and exit
  --columns STRING      A list of columns that you would like returned. Def. =
                        "*"
  --filter STRING       Restrictions to apply to variants (SQL syntax)
  --min-kindreds MIN_KINDREDS
                        The min. number of kindreds that must have a candidate
                        variant in a gene.
  --families FAMILIES   Restrict analysis to a specific set of 1 or more
                        (comma) separated) families
  -d MIN_SAMPLE_DEPTH   The minimum aligned sequence depth required for
                        each sample in a family (default = 0)
  --gt-pl-max GT_PHRED_LL
                        The maximum phred-scaled genotype likelihod (PL)
                        allowed for each sample in a family.
  --allow-unaffected    consider candidates that also appear in unaffected samples.
```

`de_novo`: Identifying potential de novo mutations.

Note: 1. This tool requires that you identify familial relationships via a PED file when loading your VCF into gemini via:

```
gemini load -v my.vcf -p my.ped my.db
```

Genotype Requirements

- all affecteds must be het
- [affected] all unaffected must be homref or homalt
- at least 1 affected kid must have unaffected parents
- [strict] if an affected has affected parents, it's not de_novo
- [strict] all affected kids must have unaffected (or no) parents
- [strict] warning if none of the affected samples have parents.

The last 3 items, prefixed with [strict] can be turned off with *–lenient*

If *–allow-unaffected* is specified, then the item prefixed [affected] is not required.

Example PED file format for GEMINI

```
#Family_ID      Individual_ID   Paternal_ID     Maternal_ID     Sex     Phenotype
↪  Ethnicity
1       S173    S238    S239    1       2       caucasian
1       S238    -9      -9      1       1       caucasian
1       S239    -9      -9      2       1       caucasian
2       S193    S230    S231    1       2       caucasian
2       S230    -9      -9      1       1       caucasian
2       S231    -9      -9      2       1       caucasian
3       S242    S243    S244    1       2       caucasian
3       S243    -9      -9      1       1       caucasian
3       S244    -9      -9      2       1       caucasian
4       S253    S254    S255    1       2       caucasianNEuropean
```

```
4       S254    -9      -9      1       1       caucasianNEuropean
4       S255    -9      -9      2       1       caucasianNEuropean
```

Assuming you have defined the familial relationships between samples when loading your VCF into GEMINI, one can leverage a built-in tool for identifying de novo (a.k.a spontaneous) mutations that arise in offspring.

example

```
$ gemini de_novo --columns "chrom,start,end" test.de_novo.db
chrom       start   end     variant_id      family_id       family_members  family_
↪genotypes        samples family_count
chr10       1142207 1142208 1       1       1_dad(dad;unaffected),1_mom(mom;
↪unaffected),1_kid(child;affected)       T/T,T/T,T/C     1_kid   1
chr10       48003991        48003992        2       2       2_dad(dad;unaffected),2_
↪mom(mom;unaffected),2_kid(child;affected)       C/C,C/C,C/T     2_kid   1
chr10       48004991        48004992        3       3       3_dad(dad;unaffected),3_
↪mom(mom;unaffected),3_kid(child;affected)       C/C,C/C,C/T     3_kid   1
chr10       135336655       135336656       4       4       1_dad(dad;unaffected),1_
↪mom(mom;unaffected),1_kid(child;affected)       G/G,G/G,G/A     1_kid   2
chr10       135336655       135336656       4       4       2_dad(dad;unaffected),2_
↪mom(mom;unaffected),2_kid(child;affected)       G/G,G/G,G/A     2_kid   2
chr10       135369531       135369532       5       5       1_dad(dad;unaffected),1_
↪mom(mom;unaffected),1_kid(child;affected)       T/T,T/T,T/C     1_kid   3
chr10       135369531       135369532       5       5       2_dad(dad;unaffected),2_
↪mom(mom;unaffected),2_kid(child;affected)       T/T,T/T,T/C     2_kid   3
chr10       135369531       135369532       5       5       3_dad(dad;unaffected),3_
↪mom(mom;unaffected),3_kid(child;affected)       T/T,T/T,T/C     3_kid   3
```

Note: The output will always start with the the requested columns followed by the 5 columns enumerated at the start of this document.

```
$ gemini de_novo -d 50 --columns "chrom,start,end" test.de_novo.db
chrom       start   end     variant_id      family_id       family_members  family_
↪genotypes        samples family_count
chr10       135369531       135369532       5       5       3_dad(dad;unaffected),3_
↪mom(mom;unaffected),3_kid(child;affected)       T/T,T/T,T/C     3_kid   1
```

example

if we wanted to restrict candidate variants to solely those with a HIGH predicted functional consequence, we could use the following:

```
$ gemini de_novo \
      --columns "chrom, start, end, ref, alt" \
      --filter "impact_severity = 'HIGH'" \
      test.de_novo.db
chrom       start   end     ref     alt     variant_id      family_id       family_
↪members  family_genotypes        samples family_count
chr10       1142207 1142208 T       C       1       1       1_dad(dad;unaffected),1_
↪mom(mom;unaffected),1_kid(child;affected)       T/T,T/T,T/C     1_kid   1
```

example

the following command would further restrict candidate genes to those genes with a de novo variant in at least two families:

```
$ gemini de_novo \
      --columns "chrom, start, end, ref, alt" \
      --filter "impact_severity = 'HIGH'" \
      --min-kindreds 2 \
      test.de_novo.db
```

example

By default, candidate de novo variants are reported for families in the database. One can restrict the analysis to variants in specific familes with the `--families` option. Families should be provided as a comma-separated list

```
$ gemini de_novo --families 1 my.db
$ gemini de_novo --families 1,7 my.db
```

`autosomal_recessive`: Find variants meeting an autosomal recessive model.

Warning: By default, this tool requires that you identify familial relationships via a PED file when loading your VCF into GEMINI. For example:

```
gemini load -v my.vcf -p my.ped my.db
```

However, in the absence of established parent/child relationships in the PED file, GEMINI will issue a WARNING, yet will attempt to identify autosomal recessive candidates for all samples marked as "affected".

Genotype Requirements

- all affecteds must be hom_alt
- [affected] no unaffected can be hom_alt (can be unknown)
- [strict] if parents exist they must be unaffected and het for all affected kids
- [strict] if there are no affecteds that have a parent, a warning is issued.

if *–lenient* is specified, the 2 points prefixed with "[strict]" are not required.

if *–allow-unaffected* is specified, the point prefix with "[affected]" is not required.

default behavior

Assuming you have defined the familial relationships between samples when loading your VCF into GEMINI, one can leverage a built-in tool for identifying variants that meet an autosomal recessive inheritance pattern. The reported variants will be restricted to those variants having the potential to impact the function of affecting protein coding transcripts.

For the following examples, let's assume we have a PED file for 3 different families as follows (the kids are affected in each family, but the parents are not):

```
$ cat families.ped
1   1_dad   0       0       -1      1
1   1_mom   0       0       -1      1
1   1_kid   1_dad   1_mom   -1      2
2   2_dad   0       0       -1      1
2   2_mom   0       0       -1      1
2   2_kid   2_dad   2_mom   -1      2
3   3_dad   0       0       -1      1
3   3_mom   0       0       -1      1
3   3_kid   3_dad   3_mom   -1      2
```

```
$ gemini autosomal_recessive test.auto_rec.db --columns "chrom,start,end,gene"
chrom       start   end     gene    variant_id      family_id       family_members␣
↪family_genotypes       samples family_count
chr10       48003991        48003992        ASAH2C  2       2       1_dad(dad;
↪unaffected),1_mom(mom;unaffected),1_kid(child;affected)       C/T,C/T,T/T     1_kid␣
↪  1
chr10       48004991        48004992        ASAH2C  3       3       2_dad(dad;
↪unaffected),2_mom(mom;unaffected),2_kid(child;affected)       C/T,C/T,T/T     2_kid␣
↪  1
chr10       135369531       135369532       SYCE1   5       5       3_dad(dad;
↪unaffected),3_mom(mom;unaffected),3_kid(child;affected)       T/C,T/C,C/C     3_kid␣
↪  1
chr10       1142207 1142208 WDR37   1       1       1_dad(dad;unaffected),1_mom(mom;
↪unaffected),1_kid(child;affected)      T/C,T/C,C/C     1_kid   2
chr10       1142207 1142208 WDR37   1       1       2_dad(dad;unaffected),2_mom(mom;
↪unaffected),2_kid(child;affected)      T/C,T/C,C/C     2_kid   2
```

Note: The output will always start with the requested columns and end with the 5 extra columns enumerated at the start of this document.

To restrict the report to genes with variants (doesn't have to be the _same_ variant) observed in at least two kindreds, use the following:

```
$ gemini autosomal_recessive \
    --columns "gene, chrom, start, end, ref, alt, impact, impact_severity" \
    --min-kindreds 2 \
    test.auto_rec.db
gene        chrom   start   end     ref     alt     impact  impact_severity variant_
↪id     family_id       family_members  family_genotypes        samples family_count
ASAH2C      chr10   48003991        48003992        C       T       non_syn_coding ␣
↪MED     2       2       1_dad(dad;unaffected),1_mom(mom;unaffected),1_kid(child;
↪affected)      C/T,C/T,T/T     1_kid   1
ASAH2C      chr10   48004991        48004992        C       T       non_syn_coding ␣
↪MED     3       3       2_dad(dad;unaffected),2_mom(mom;unaffected),2_kid(child;
↪affected)      C/T,C/T,T/T     2_kid   1
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪       1_dad(dad;unaffected),1_mom(mom;unaffected),1_kid(child;affected)       T/C,T/
↪C,C/C      1_kid   2
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪       2_dad(dad;unaffected),2_mom(mom;unaffected),2_kid(child;affected)       T/C,T/
↪C,C/C      2_kid   2
```

to report only those with a HIGH predicted functional consequence, we could use the following:

```
$ gemini autosomal_recessive \
    --columns "gene, chrom, start, end, ref, alt, impact, impact_severity" \
    --min-kindreds 2 \
    --filter "impact_severity = 'HIGH'" \
    test.auto_rec.db
gene        chrom   start   end     ref     alt     impact  impact_severity variant_
↪id      family_id       family_members  family_genotypes        samples family_count
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪      1_dad(dad;unaffected),1_mom(mom;unaffected),1_kid(child;affected)       T/C,T/
↪C,C/C      1_kid   2
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪      2_dad(dad;unaffected),2_mom(mom;unaffected),2_kid(child;affected)       T/C,T/
↪C,C/C      2_kid   2
```

To limit to confidently called genotypes:

```
$ gemini autosomal_dominant \
    --columns "gene, chrom, start, end, ref, alt, impact, impact_severity" \
    --filter "impact_severity = 'HIGH'" \
    --min-kindreds 1 \
    --gt-pl-max 10 \
    my.db
```

`autosomal_dominant`: Find variants meeting an autosomal dominant model.

Warning: 0. version 0.16.0 changes the behavior of this tool to be more strict. To regain more lenient behavior, specify –lenient and –allow-unaffected.

By default, this tool requires that you identify familial relationships via a PED file when loading your VCF into GEMINI. For example:

```
gemini load -v my.vcf -p my.ped my.db
```

Genotype Requirements

- All affecteds must be het
- [affected] No unaffected can be het or homalt (can be unknown)
- de_novo mutations are not auto_dom (at least not in the first generation)
- At least 1 affected must have 1 affected parent (or have no parents).
- If no affected has a parent, a warning is issued.
- [strict] All affecteds must have parents with known phenotype.
- [strict] All affected kids must have at least 1 affected parent

If *–lenient* is specified, the items prefixed with "[strict]" are not required.

If *–allow-unaffected* is specified, the item prefix with "[affected]" is not required.

Note that for autosomal dominant *–lenient* allows singleton affecteds to be used to meet the *–min-kindreds* requirement if they are HET.

If there is incomplete penetrance in the kindred (unaffected obligate carriers), these individuals currently must be coded as having unknown phenotype or as being affected.

default behavior

For the following examples, let's assume we have a PED file for 3 different families as follows (the kids are affected in each family, but the parents are not):

```
$ cat families.ped
1   1_dad   0       0       -1      1
1   1_mom   0       0       -1      1
1   1_kid   1_dad   1_mom   -1      2
2   2_dad   0       0       -1      1
2   2_mom   0       0       -1      2
2   2_kid   2_dad   2_mom   -1      2
3   3_dad   0       0       -1      2
3   3_mom   0       0       -1      -9
3   3_kid   3_dad   3_mom   -1      2
```

```
$ gemini autosomal_dominant test.auto_dom.db --columns "chrom,start,end,gene"
chrom       start   end     gene    variant_id      family_id       family_members␣
↪family_genotypes       samples family_count
chr10       48003991        48003992        ASAH2C  3       3       2_dad(dad;
↪unaffected),2_mom(mom;affected),2_kid(child;affected) C/C,C/T,C/T     2_mom,2_kid␣
↪  2
chr10       48004991        48004992        ASAH2C  4       4       2_dad(dad;
↪unaffected),2_mom(mom;affected),2_kid(child;affected) C/C,C/T,C/T     2_mom,2_kid␣
↪  2
chr10       48003991        48003992        ASAH2C  3       3       3_dad(dad;
↪affected),3_mom(mom;unknown),3_kid(child;affected)    C/T,C/C,C/T     3_dad,3_kid␣
↪  2
chr10       48004991        48004992        ASAH2C  4       4       3_dad(dad;
↪affected),3_mom(mom;unknown),3_kid(child;affected)    C/T,C/C,C/T     3_dad,3_kid␣
↪  2
chr10       135336655       135336656       SPRN    5       5       3_dad(dad;
↪affected),3_mom(mom;unknown),3_kid(child;affected)    G/A,G/G,G/A     3_dad,3_kid␣
↪  1
chr10       1142207 1142208 WDR37   1       1       2_dad(dad;unaffected),2_mom(mom;
↪affected),2_kid(child;affected) T/T,T/C,T/C     2_mom,2_kid     2
chr10       1142207 1142208 WDR37   1       1       3_dad(dad;affected),3_mom(mom;
↪unknown),3_kid(child;affected)    T/C,T/T,T/C     3_dad,3_kid     2
```

```
$ gemini autosomal_dominant \
    --columns "gene, chrom, start, end, ref, alt, impact, impact_severity" \
    --min-kindreds 2 \
    test.auto_dom.db
gene        chrom   start   end     ref     alt     impact  impact_severity variant_
↪id      family_id       family_members  family_genotypes        samples family_count
ASAH2C      chr10   48003991        48003992        C       T       non_syn_coding␣
↪MED     3       3       2_dad(dad;unaffected),2_mom(mom;affected),2_kid(child;
↪affected) C/C,C/T,C/T     2_mom,2_kid     2
ASAH2C      chr10   48004991        48004992        C       T       non_syn_coding␣
↪MED     4       4       2_dad(dad;unaffected),2_mom(mom;affected),2_kid(child;
↪affected) C/C,C/T,C/T     2_mom,2_kid     2
ASAH2C      chr10   48003991        48003992        C       T       non_syn_coding␣
↪MED     3       3       3_dad(dad;affected),3_mom(mom;unknown),3_kid(child;
↪affected)    C/T,C/C,C/T     3_dad,3_kid     2
ASAH2C      chr10   48004991        48004992        C       T       non_syn_coding␣
↪MED     4       4       3_dad(dad;affected),3_mom(mom;unknown),3_kid(child;
↪affected)    C/T,C/C,C/T     3_dad,3_kid     2
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪      2_dad(dad;unaffected),2_mom(mom;affected),2_kid(child;affected) T/T,T/C,T/C␣
↪  2_mom,2_kid     2
```

```
WDR37       chr10   1142207 1142208 T       C       stop_loss       HIGH    1       1␣
↪       3_dad(dad;affected),3_mom(mom;unknown),3_kid(child;affected)    T/C,T/T,T/C  ␣
↪  3_dad,3_kid     2
```

`x_linked_recessive`: x-linked recessive inheritance

Note that as of version 0.19.0, we do not account for the pseudo autosomal regions. The 'X' chromosome can be specifing using –X (defaults to 'chrX' and 'X')

Genotype Requirements

- Affected females must be HOM_ALT
- Unaffected females are HET or HOM_REF
- Affected males are not HOM_REF
- Unaffected males are HOM_REF

`x_linked_dominant`: x-linked dominant inheritance

Note that as of version 0.19.0, we do not account for the pseudo autosomal regions. The 'X' chromosome can be specifing using –X (defaults to 'chrX' and 'X')

Genotype Requirements

- Affected males are HET or HOM_ALT
- Affected females must be HET
- Unaffecteds must be HOM_REF
- girls of affected dad must be affected
- boys of affected dad must be unaffected
- mothers of affected males must be het (and affected) [added in 0.19.1]
- at least 1 parent of affected females must be het (and affected). [added in 0.19.1]

`x_linked_de_novo`: x-linked de novo

Note that as of version 0.19.0, we do not account for the pseudo autosomal regions. The 'X' chromosome can be specifing using –X (defaults to 'chrX' and 'X')

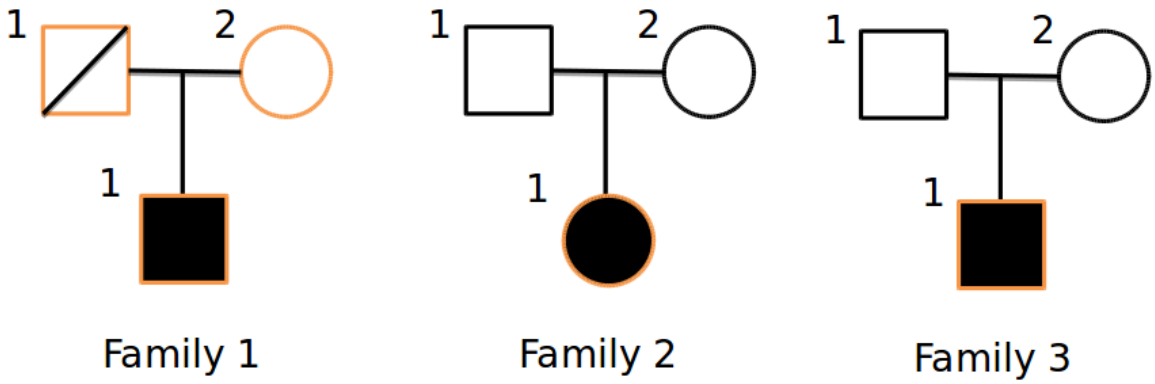
Genotype Requirements

- affected female child must be het
- affected male child must be hom_alt (or het)
- parents should be unaffected and hom_ref

`gene_wise`: Custom genotype filtering by gene.

The gemini query tool allows querying by variant and the inheritance tools described above enable querying by gene for fixed inheritance patterns. The *gene_wise* tool allows querying by gene with custom genotype filters to bridge the gap between these tools.

With this tool, multiple *–gt-filter* s can be specified. Each filter can be any valid filter; often, it will make sense to have 1 filter for each family. For example, given this pedigree:



Where only the orange samples are sequenced, we could devise a query:

```
gemini gene_wise $db \
    --min-filters 3 \
    --gt-filter "gt_types.fam1_kid == HET and gt_types.fam1_mom == HOM_REF and gt_
↪types.fam1_dad == HOM_REF" \
    --gt-filter "gt_types.fam2_kid == HET" \
    --gt-filter "gt_types.fam3_kid == HET" \
    --columns "chrom,start,end,gene,impact,impact_severity" \
    --filter "max_aaf_all < 0.005"
```

The *–min-filters* option means that we want all 3 of those filters to be met in a gene in order for variants in that gene to be reported. We can envision a scenario where we have 6 families (and 6 filters) and we want to report genes where 4 of them meet the filters. In that case, the query would have 6 *–gt-filter* s and *–min-filters* of 3.

This differs from using gemini query with a single *–gt-filter* that combines each of those terms with an *and* because this allows each filter to be met **in a different variant** but **in the same gene** while the gemini query tool applies all elements of the single filter to each variant.

The output from the above query is:

```
chrom  start     end       gene      impact              impact_severity  variant_
↪filters  n_gene_variants  gene_filters
chr5   60839982  60839983  ZSWIM6    non_syn_coding      MED              1,2,3
↪      1                 1,2,3
chr6   32548031  32548032  HLA-DRB1  non_syn_coding      MED              1
↪      4                 1,2,3
chr6   32552059  32552060  HLA-DRB1  frame_shift         HIGH             2
↪      4                 1,2,3
chr6   32552131  32552132  HLA-DRB1  inframe_codon_gain  MED              3
↪      4                 1,2,3
chr6   32552136  32552137  HLA-DRB1  non_syn_coding      MED              3
↪      4                 1,2,3
```

Note that the first gene has the same variant for all 3 families, so we could have found this with the gemini query tool. However, for the HLA gene, each of the 3 filters passed in different variant so this would be missed by the query tool which only looks at a single variant at a time.

As with the other tools, this tool orders by chromosome and gene and it applies *WHERE (is_exonic = 1 AND impact_severity != 'LOW')"* to the query.

- The *variant_filters* column shows which filters were passed by the variant.
- The *n_gene_variants* column shows how many variants in the gene are being reported.
- The *gene_filter* column shows which filters in the gene passed by any variant.

Multiple *–gt-filter-required* filters can also be specified. Each filter added to this argument is required to pass for each variant and it does not contribute to the *–min-filters* argument. This can be used with, or instead of *–gt-filter* E.g.

```
gemini gene_wise  \
    --columns "gene, chrom, start, end, ref, alt, impact, impact_severity" \
    --gt-filter-required "((gt_depths).(*).(>10).(all))" \
    --gt-filter "gt_types.fam1_kid == HET and gt_types.fam1_mom == HOM_REF and gt_
↪types.fam1_dad == HOM_REF" \
    --gt-filter "gt_types.fam2_kid == HET" \
    --gt-filter "gt_types.fam3_kid == HET" \
    --min-filters 2 \
    test.db
```

will required that all samples meet the minimum depth filter and then keep the subset of those that meet 2 out of the 3 *–gt-filters*.

`--where`

By default *gene_wise* limits to variants passing the clause:

is_exonic = 1 AND impact_severity != 'LOW'

but this can be changed with the –where clause, e.g.:

```
--where "(is_exonic = 1 or is_splicing = 1) AND impact_severity != 'LOW'"
```

`pathways`: Map genes and variants to KEGG pathways.

Mapping genes to biological pathways is useful in understanding the function/role played by a gene. Likewise, genes involved in common pathways is helpful in understanding heterogeneous diseases. We have integrated the KEGG pathway mapping for gene variants, to explain/annotate variation. This requires your VCF be annotated with either snpEff/VEP.

Examples:

```
$ gemini pathways -v 68 example.db
chrom   start   end     ref     alt     impact  sample  genotype        gene    ␣
↪transcript      pathway
chr10   52004314        52004315        T       C       intron  M128215 C/C     ASAH2␣
↪  ENST00000395526 hsa00600:Sphingolipid_metabolism,hsa01100:Metabolic_pathways
```

```
chr10   126678091       126678092       G       A       stop_gain       M128215 G/A
↪  CTBP2   ENST00000531469 hsa05220:Chronic_myeloid_leukemia,hsa04310:Wnt_signaling_
↪pathway,hsa04330:Notch_signaling_pathway,hsa05200:Pathways_in_cancer
chr16   72057434        72057435        C       T       non_syn_coding  M10475  C/T
↪  DHODH   ENST00000219240 hsa01100:Metabolic_pathways,hsa00240:Pyrimidine_metabolism
```

Here, -v specifies the version of the Ensembl genes used to build the KEGG pathway map. Hence, use versions that match the VEP/snpEff versions of the annotated vcf for correctness. For e.g VEP v2.6 and snpEff v3.1 use Ensembl 68 version of the genomes.

We currently support versions 66 through 71 of the Ensembl genes

`--lof`

By default, all gene variants that map to pathways are reported. However, one may want to restrict the analysis to LoF variants using the `--lof` option.

```
$ gemini pathways --lof -v 68 example.db
chrom   start   end     ref     alt     impact  sample  genotype        gene
↪transcript      pathway
chr10   126678091       126678092       G       A       stop_gain       M128215 G/A
↪  CTBP2   ENST00000531469 hsa05220:Chronic_myeloid_leukemia,hsa04310:Wnt_signaling_
↪pathway,hsa04330:Notch_signaling_pathway,hsa05200:Pathways_in_cancer
```

`interactions`: Find genes among variants that are interacting partners.

Integrating the knowledge of the known protein-protein interactions would be useful in explaining variation data. Meaning to say that a damaging variant in an interacting partner of a potential protein may be equally interesting as the protein itself. We have used the HPRD binary interaction data to build a p-p network graph which can be explored by GEMINI.

Examples:

```
$ gemini interactions -g CTBP2 -r 3 example.db
sample  gene    order_of_interaction    interacting_gene
M128215 CTBP2   0_order:        CTBP2
M128215 CTBP2   1_order:        RAI2
M128215 CTBP2   2_order:        RB1
M128215 CTBP2   3_order:        TGM2,NOTCH2NL
```

Return CTBP2 (-g) interacting gene variants till the third order (-r)

`lof_interactions`

Use this option to restrict your analysis to only LoF variants.

```
$ gemini lof_interactions -r 3 example.db
sample  lof_gene        order_of_interaction    interacting_gene
M128215 TGM2    1_order:        RB1
M128215 TGM2    2_order:        none
M128215 TGM2    3_order:        NOTCH2NL,CTBP2
```

Meaning to say return all LoF gene TGM2 (in sample M128215) interacting partners to a 3rd order of interaction.

--var

An extended variant information (chrom, start, end etc.) for the interacting gene may be achieved with the –var option for both the `interactions` and the `lof_interactions`

```
$ gemini interactions -g CTBP2 -r 3 --var example.db
sample  gene    order_of_interaction    interacting_gene        var_id  chrom   start
↪  end     impact  biotype in_dbsnp        clinvar_sig     clinvar_disease_name
↪aaf_1kg_all     aaf_esp_all
M128215 CTBP2   0       CTBP2   5       chr10   126678091       126678092       stop_
↪gain       protein_coding  1       None    None    None    None
M128215 CTBP2   1       RAI2    9       chrX    17819376        17819377        non_
↪syn_coding  protein_coding  1       None    None    1       0.000473
M128215 CTBP2   2       RB1     7       chr13   48873834        48873835
↪upstream        protein_coding  1       None    None    0.94    None
M128215 CTBP2   3       NOTCH2NL        1       chr1    145273344       145273345
↪  non_syn_coding  protein_coding  1       None    None    None    None
M128215 CTBP2   3       TGM2    8       chr20   36779423        36779424        stop_
↪gain       protein_coding  0       None    None    None    None
```

```
$ gemini lof_interactions -r 3 --var example.db
sample  lof_gene        order_of_interaction    interacting_gene        var_id  chrom
↪  start   end     impact  biotype in_dbsnp        clinvar_sig     clinvar_disease_
↪name    aaf_1kg_all     aaf_esp_all
M128215 TGM2    1       RB1     7       chr13   48873834        48873835
↪upstream        protein_coding  1       None    None    0.94    None
M128215 TGM2    3       NOTCH2NL        1       chr1    145273344       145273345
↪  non_syn_coding  protein_coding  1       None    None    None    None
M128215 TGM2    3       CTBP2   5       chr10   126678091       126678092       stop_
↪gain       protein_coding  1       None    None    None    None
```

`lof_sieve`: Filter LoF variants by transcript position and type

Not all candidate LoF variants are created equal. For e.g, a nonsense (stop gain) variant impacting the first 5% of a polypeptide is far more likely to be deleterious than one affecting the last 5%. Assuming you've annotated your VCF with snpEff v3.0+, the lof_sieve tool reports the fractional position (e.g. 0.05 for the first 5%) of the mutation in the amino acid sequence. In addition, it also reports the predicted function of the transcript so that one can segregate candidate LoF variants that affect protein_coding transcripts from processed RNA, etc.

```
$ gemini lof_sieve chr22.low.exome.snpeff.100samples.vcf.db
chrom   start   end ref alt highest_impact  aa_change   var_trans_pos   trans_aa_
↪length var_trans_pct   sample  genotype    gene    transcript  trans_type
chr22   17072346    17072347    C   T   stop_gain   W365*   365 557 0.655296229803
↪NA19327 C|T CCT8L2  ENST00000359963 protein_coding
chr22   17072346    17072347    C   T   stop_gain   W365*   365 557 0.655296229803
↪NA19375 T|C CCT8L2  ENST00000359963 protein_coding
chr22   17129539    17129540    C   T   splice_donor    None    None    None    None
↪  NA18964 T|C TPTEP1  ENST00000383140 lincRNA
chr22   17129539    17129540    C   T   splice_donor    None    None    None    None
↪  NA19675 T|C TPTEP1  ENST00000383140 lincRNA
```

`amend`: updating / changing the sample information

Occassionally one may need to update the sample information stored in the `samples` table. The `amend` tool allows one to provide an updated PED file as input and it will update each sample_id in the PED file that matches a sample_id.

For example, assume you have already loaded a GEMINI database with a `samples.ped` where mom and dad are unaffected and kid isaffected:

```
$ cat samples.ped
1 dad 0 0 1  1
1 mom 0 0 1  1
1 kid dad mom 2  2

$ gemini load -v my.vcf -p samples.ped -t VEP my.db
```

Now, let's say you realized that the dad is also affected and you want to correct the `samples` table accordingly. You would first edit the PED file and then run the `amend` tool using the updated PED.

```
$ cat samples.ped
1 dad 0 0 1  2
1 mom 0 0 1  1
1 kid dad mom 2  2

$ gemini amend --sample samples.ped my.db
```

`annotate`: adding your own custom annotations

It is inevitable that researchers will want to enhance the gemini framework with their own, custom annotations. `gemini` provides a sub-command called `annotate` for exactly this purpose. As long as you provide a `tabix`'ed annotation file in BED or VCF format, the `annotate` tool will, for each variant in the variants table, screen for overlaps in your annotation file and update a one or more new column in the variants table that you may specify on the command line. This is best illustrated by example.

Let's assume you have already created a gemini database of a VCF file using the `load` module.

```
$ gemini load -v my.vcf -t snpEff my.db
```

Now, let's imagine you have an annotated file in BED format (`important.bed`) that describes regions of the genome that are particularly relevant to your lab's research. You would like to annotate in the gemini database which variants overlap these crucial regions. We want to store this knowledge in a new column in the `variants` table called `important_variant` that tracks whether a given variant overlapped (1) or did not overlap (0) intervals in your annotation file.

To do this, you must first TABIX your BED file:

```
$ bgzip important.bed
$ tabix -p bed important.bed.gz
```

`-a boolean` Did a variant overlap a region or not?

Note: Formerly, the `-a` option was the `-t` option.

Now, you can use this TABIX'ed file to annotate which variants overlap your important regions. In the example below, the results will be stored in a new column called "important". The `-t boolean` option says that you just want to track whether (1) or not (0) the variant overlapped one or more of your regions.

```
$ gemini annotate -f important.bed.gz -c important -a boolean my.db
```

Since a new columns has been created in the database, we can now directly query the new column. In the example results below, the first and third variants overlapped a crucial region while the second did not.

```
$ gemini query \
    -q "select chrom, start, end, variant_id, important from variants" \
    my.db \
    | head -3
chr22   100    101    1   1
chr22   200    201    2   0
chr22   300    500    3   1
```

`-a count` How many regions did a variant overlap?

Instead of a simple yes or no, we can use the `-t count` option to *count* how many important regions a variant overlapped. It turns out that the 3rd variant actually overlapped two important regions.

```
$ gemini annotate -f important.bed.gz -c important -a count my.db

$ gemini query \
    -q "select chrom, start, end, variant_id, crucial from variants" \
    my.db \
    | head -3
chr22   100    101    1   1
chr22   200    201    2   0
chr22   300    500    3   2
```

`-a extract` Extract specific values from a BED file

Lastly, we may also extract values from specific fields in a BED file (or from the INFO field in a VCF) and populate one or more new columns in the database based on overlaps with the annotation file and the values of the fields therein. To do this, we use the `-a extract` option.

This is best described with an example. To set this up, let's imagine that we have a VCF file from a different experiment and we want to annotate the variants in our GEMINI database with the allele frequency and depth tags from the INFO fields for the same variants in this other VCF file.

> # bgzip and tabix the vcf for use with the annotate tool. $ bgzip other.vcf $ tabix other.vcf.gz

Now that we have a proper TABIX'ed VCF file, we can use the `-a extract` option to populate new columns in the GEMINI database. In order to do so, we must specify:

1. its type (e.g., text, int, float,) (`-t`)
2. the field in the INFO column of the VCF file that we should use to extract data with which to populate the new column (`-e`)
3. what operation should be used to summarize the data in the event of multiple overlaps in the annotation file (`-o`)
4. (optionally) the name of the column we want to add (`-c`), if this is not specified, it will use the value from `-e`.

For example, let's imagine we want to create a new column called "other_allele_freq" using the AF field in our VCF file to populate it.

```
$ gemini annotate -f other.vcf.gz \
                  -a extract \
                  -c other_allele_freq \
                  -t float \
                  -e AF \
                  -o mean \
                  my.db
```

This create a new column in `my.db` called `other_allele_freq` and this new column will be a FLOAT. In the event of multiple records in the VCF file overlapping a variant in the database, the average (mean) of the allele frequencies values from the VCF file will be used.

At this point, one can query the database based on the values of the new `other_allele_freq` column:

```
$ gemini query -q "select * from variants where other_allele_freq < 0.01" my.db
```

`-t TYPE` Specifying the column type(s) when using `-a extract`

The `annotate` tool will create three different types of columns via the `-t` option:

1. Floating point columns for annotations with decimal precision as above (`-t float`)
2. Integer columns for integral annotations (`-t integer`)
3. Text columns for string columns such as "valid", "yes", etc. (`-t text`)

Note: The `-t` option is only valid when using the `-a extract` option.

`-o OPERATION` Specifying the summary operations when using `-a extract`

In the event of multiple overlaps between a variant and records in the annotation file, the `annotate` tool can summarize the values observed with multiple options:

1. `-o mean`. Compute the average of the values. **They must be numeric**.
2. `-o median`. Compute the median of the values. **They must be numeric**.
3. `-o min`. Compute the minimum of the values. **They must be numeric**.
4. `-o max`. Compute the maximum of the values. **They must be numeric**.
5. `-o mode`. Compute the maximum of the values. **They must be numeric**.
6. `-o first`. Use the value from the **first** record in the annotation file.
7. `-o last`. Use the value from the **last** record in the annotation file.
8. `-o list`. Create a comma-separated list of the observed values. **-t must be text**
9. `-o uniq_list`. Create a comma-separated list of the **distinct** (i.e., non-redundant) observed values. **-t must be text**
10. `-o sum`. Compute the sum of the values. **They must be numeric**.

Note: The `-o` option is only valid when using the `-a extract` option.

Annotating with VCF

Most of the examples to this point have pulled a column from a *tabix* indexed bed file. It is likewise possible to pull from the INFO field of a *tabix* index VCF. The syntax is identical but the `-e` operation will specify the names of fields in the INFO column to pull. By default, those names will be used, but that can still be specified with the *-c* column. Here are some example uses

```
# put a DP column in the db:
gemini annotate -f anno.vcf.gz -o list -e DP -t integer my.db

# ... and name it 'depth'
gemini annotate -f anno.vcf.gz -o list -e DP -c depth -t integer my.db

# use multiple columns
gemini annotate -f anno.vcf.gz -o list,mean -e DP,Qmeter -c depth,qmeter -t integer␣
↪my.db
```

Missing values are allowed since we expect that in some cases an annotation VCF will not have all INFO fields specified for all variants.

Note: We recommend decomposing and normalizing variants before annotating. See *Step 1. split, left-align, and trim variants* for a detailed explanation of how to do this.

Extracting and populating multiple columns at once.

One can also extract and populate multiple columns at once by providing comma-separated lists (no spaces) of column names (`-c`), types (`-t`), numbers (`-e`), and summary operations (`-o`). For example, recall that in the VCF example above, we created a TABIX'ed BED file containg the allele frequency and depth values from the INFO field as the 4th and 5th columns in the BED, respectively.

Instead of running the `annotate` tool twice (once for eaxh column), we can run the tool once and load both columns in the same run. For example:

```
$ gemini annotate -f other.bed.gz \
                  -a extract \
                  -c other_allele_freq,other_depth \
                  -t float,integer \
                  -e 4,5 \
                  -o mean,max \
                  my.db
```

We can then use each of the new columns to filter variants with a GEMINI query:

```
$ gemini query -q "select * from variants \
                   where other_allele_freq < 0.01 \
                   and other_depth > 100" my.db
```

region: Extracting variants from specific regions or genes

One often is concerned with variants found solely in a particular gene or genomic region. gemini allows one to extract variants that fall within specific genomic coordinates as follows:

--reg

```
$ gemini region --reg chr1:100-200 my.db
```

--gene

Or, one can extract variants based on a specific gene name.

```
$ gemini region --gene PTPN22 my.db
```

--columns

By default, this tool reports all columns in the variants table. One may choose to report only a subset of the columns using the --columns option. For example, to report just the gene, chrom, start, end, ref, alt, impact, and impact_severity columns, one would use the following:

```
$ gemini region --gene DHODH \
                --columns "chrom, start, end, ref, alt, gene, impact" \
                my.db

chr16   72057281    72057282    A   G   DHODH   intron
chr16   72057434    72057435    C   T   DHODH   non_syn_coding
chr16   72059268    72059269    T   C   DHODH   downstream
```

--filter

By default, this tool will report all variants regardless of their putative functional impact. In order to apply additional constraints on the variants returned, one can use the --filter option. Using SQL syntax, conditions applied with the ‘‘–filter option become WHERE clauses in the query issued to the GEMINI database. For example, if we wanted to restrict candidate variants to solely those with a HIGH predicted functional consequence, we could use the following:

```
$ gemini region --gene DHODH \
                --columns "chrom, start, end, ref, alt, gene, impact" \
                --filter "alt='G'"
                my.db

chr16   72057281    72057282    A   G   DHODH   intron
```

--json

Reporting query output in JSON format may enable HTML/Javascript apps to query GEMINI and retrieve the output in a format that is amenable to web development protocols.

To report in JSON format, use the --json option. For example:

```
$ gemini region --gene DHODH \
                --columns "chrom, start, end, ref, alt, gene, impact" \
                --filter "alt='G'"
                --json
                my.db

{"chrom": "chr16", "start": 72057281, "end": 72057282, "ref": "A", "alt": "G", "gene
↪": "DHODH"}
```

`windower`: Conducting analyses on genome "windows".

`gemini` includes a convenient tool for computing variation metrics across genomic windows (both fixed and sliding). Here are a few examples to whet your appetite. If you're still hungry, contact us.

Compute the average nucleotide diversity for all variants found in non-overlapping, 50Kb windows.

```
$ gemini windower -w 50000 -s 0 -t nucl_div -o mean my.db
```

Compute the average nucleotide diversity for all variants found in 50Kb windows that overlap by 10kb.

```
$ gemini windower -w 50000 -s 10000 -t nucl_div -o mean my.db
```

Compute the max value for HWE statistic for all variants in a window of size 10kb

```
$ gemini windower  -w 10000 -t hwe -o max my.db
```

`stats`: Compute useful variant statistics.

The `stats` tool computes some useful variant statistics like

Compute the transition and transversion ratios for the snps

```
$ gemini stats --tstv my.db
ts      tv      ts/tv
4       5       0.8
```

`--tstv-coding`

Compute the transition/transversion ratios for the snps in the coding regions.

`--tstv-noncoding`

Compute the transition/transversion ratios for the snps in the non-coding regions.

Compute the type and count of the snps.

```
$ gemini stats --snp-counts my.db
type    count
A->G    2
C->T    1
G->A    1
```

Calculate the site frequency spectrum of the variants.

```
$ gemini stats --sfs my.db
aaf     count
0.125   2
0.375   1
```

Compute the pair-wise genetic distance between each sample

```
$ gemini stats --mds my.db
sample1 sample2 distance
M10500  M10500  0.0
M10475  M10478  1.25
M10500  M10475  2.0
M10500  M10478  0.5714
```

Return a count of the types of genotypes per sample

```
$ gemini stats --gts-by-sample my.db
sample  num_hom_ref     num_het num_hom_alt     num_unknown     total
M10475  4       1       3       1       9
M10478  2       2       4       1       9
```

Return the total variants per sample (sum of homozygous and heterozygous variants)

```
$ gemini stats --vars-by-sample my.db
sample  total
M10475  4
M10478  6
```

`--summarize`

If none of these tools are exactly what you want, you can summarize the variants per sample of an arbitrary query using the –summarize flag. For example, if you wanted to know, for each sample, how many variants are on chromosome 1 that are also in dbSNP:

```
$ gemini stats --summarize "select * from variants where in_dbsnp=1 and chrom='chr1'"␣
↪my.db
sample  total   num_het num_hom_alt
M10475  1       1       0
M128215 1       1       0
M10478  2       2       0
M10500  2       1       1
```

`burden`: perform sample-wise gene-level burden calculations

The `burden` tool provides a set of utilities to perform burden summaries on a per-gene, per sample basis. By default, it outputs a table of gene-wise counts of all high impact variants in coding regions for each sample:

```
$ gemini burden test.burden.db
gene    M10475  M10478  M10500  M128215
WDR37   2       2       2       2
CTBP2   0       0       0       1
DHODH   1       0       0       0
```

--nonsynonymous

If you want to be a little bit less restrictive, you can include all non-synonymous variants instead:

```
$ gemini burden --nonsynonymous test.burden.db
gene    M10475  M10478  M10500  M128215
SYCE1   0       1       1       0
WDR37   2       2       2       2
CTBP2   0       0       0       1
ASAH2C  2       1       1       0
DHODH   1       0       0       0
```

--calpha

If your database has been loaded with a PED file describing case and control samples, you can calculate the c-alpha statistic for cases vs. control:

```
$ gemini burden --calpha test.burden.db
gene    T       c       Z       p_value
SYCE1   -0.5    0.25    -1.0    0.841344746069
WDR37   -1.0    1.5     -0.816496580928 0.792891910879
CTBP2   0.0     0.0     nan     nan
ASAH2C  -0.5    0.75    -0.57735026919  0.718148569175
DHODH   0.0     0.0     nan     nan
```

To calculate the P-value using a permutation test, use the `--permutations` option, specifying the number of permutations of the case/control labels you want to use.

--min-aaf and --max-aaf for --calpha

By default, all variants affecting a given gene will be included in the C-alpha computation. However, one may establish alternate allele frequency boundaries for the variants included using the `--min-aaf` and `--max-aaf` options.

```
$ gemini burden --calpha test.burden.db --min-aaf 0.0 --max-aaf 0.01
```

--cases and --controls for ``--calpha

If you do not have a PED file loaded, or your PED file does not follow the standard PED phenotype encoding format you can still perform the c-alpha test, but you have to specify which samples are the control samples and which are the case samples:

```
$ gemini burden --controls M10475 M10478 --cases M10500 M128215 --calpha test.burden.
↪db
gene    T       c       Z       p_value
SYCE1   -0.5    0.25    -1.0    0.841344746069
WDR37   -1.0    1.5     -0.816496580928 0.792891910879
CTBP2   0.0     0.0     nan     nan
ASAH2C  -0.5    0.75    -0.57735026919  0.718148569175
DHODH   0.0     0.0     nan     nan
```

`--nonsynonymous --calpha`

If you would rather consider all nonsynonymous variants for the C-alpha test rather than just the medium and high impact variants, add the `--nonsynonymous` flag.

`roh`: Identifying runs of homozygosity

Runs of homozygosity are long stretches of homozygous genotypes that reflect segments shared identically by descent and are a result of consanguinity or natural selection. Consanguinity elevates the occurrence of rare recessive diseases (e.g. cystic fibrosis) that represent homozygotes for strongly deleterious mutations. Hence, the identification of these runs holds medical value.

The 'roh' tool in GEMINI returns runs of homozygosity identified in whole genome data. The tool basically looks at every homozygous position on the chromosome as a possible start site for the run and looks for those that could give rise to a potentially long stretch of homozygous genotypes.

For e.g. for the given example allowing `1 HET` genotype (h) and `2 UKW` genotypes (u) the possible roh runs (H) would be:

```
genotype_run = H H H H h H H H H u H H H H H u H H H H H H H h H H H H H h H H H H H
roh_run1     = H H H H h H H H H u H H H H H u H H H H H H H
roh_run2     =           H H H H u H H H H H u H H H H H H H h H H H H H
roh_run3     =                     H H H H H u H H H H H H H h H H H H H
roh_run4     =                               H H H H H H H h H H H H H
```

roh returned for –min-snps = 20 would be:

```
roh_run1     = H H H H h H H H H u H H H H H u H H H H H H H
roh_run2     =           H H H H u H H H H H u H H H H H H H h H H H H H
```

As you can see, the immediate homozygous position right of a break (h or u) would be the possible start of a new roh run and genotypes to the left of a break are pruned since they cannot be part of a longer run than we have seen before.

Return `roh` with minimum of 50 snps, a minimum run length of 1 mb and a minimum sample depth of 20 for sample S138 (with default values for allowed number of HETS, UNKS and total depth).

```
$ gemini roh --min-snps 50 \
           --min-gt-depth 20 \
                  --min-size 1000000 \
                  -s S138 \
                  roh_run.db
chrom   start   end     sample  num_of_snps     density_per_kb  run_length_in_bp
chr2 233336080 234631638 S138 2583 1.9953 1295558
chr2    238341281       239522281       S138    2899    2.4555  1181000
```

`set_somatic`: Flag somatic variants

Somatic mutations in a tumor-normal pair are variants that are present in the tumor but not in the normal sample.

Note: 1. This tool requires that you specify the sample layout via a PED file when loading your VCF into GEMINI via:

```
gemini load -v my.vcf -p my.ped my.db
```

Example PED file format for GEMINI

```
#Family_ID      Individual_ID   Paternal_ID     Maternal_ID     Sex     Phenotype
↪  Ethnicity
1       Normal  -9      -9      0       1       -9
1       Tumor   -9      -9      0       2       -9
```

default behavior

By default, `set_somatic` simply marks variants that are genotyped as homozygous reference in the normal sample and non-reference in the tumor. More stringent somatic filtering criteria are available through tunable command line parameters.

```
$ gemini set_somatic \
    --min-depth 30 \
    --min-qual 20 \
    --min-somatic-score 18 \
    --min-tumor-depth 10 \
    --min-norm-depth 10 \
    tumor_normal.db
tum_name        tum_gt  tum_alt_freq    tum_alt_depth   tum_depth       nrm_name
↪  nrm_gt  nrm_alt_freq    nrm_alt_depth   nrm_depth       chrom   start   end
↪ref     alt     gene
tumor   GAAAAAAAAAAAAAGGTGAAAATT/GAAAAAAAAAAAAAGGTGAAAATT       0.217391304348  5
↪  23      normal  GAAAAAAAAAAAAAGGTGAAAATT/GAAAAAAAAAAAAAGGTGAAAATT       0.0     0
↪      25      chrX    132838304       132838328       GAAAAAAAAAAAAAGGTGAAAATT
↪  GAAAAAAAAAAAAAGGTGAAAATT GPC3
tumor   CTGCTATTTTG/CG  0.22    11      50      normal  CTGCTATTTTG/CTGCTATTTTG 0.0
↪  0       70      chr17   59861630        59861641        CTGCTATTTTG     CG
↪BRIP1
tumor   C/A     0.555555555556  10      18      normal  C/C     0.0     0       17
↪  chr17   7578460 7578461 C       A       TP53
tumor   C/T     0.1875  12      64      normal  C/C     0.0     0       30      chr2
↪  128046288       128046289       C       T       ERCC3
Identified and set 4 somatic mutations
```

--min-depth [None]

The minimum required combined depth for tumor and normal samples.

--min-qual [None]

The minimum required variant quality score.

--min-somatic-score [None]

The minimum required somatic score (SSC). This score is produced by various somatic variant detection algorithms including SpeedSeq, SomaticSniper, and VarScan 2.

--max-norm-alt-freq [None]

The maximum frequency of the alternate allele allowed in the normal sample.

--max-norm-alt-count [None]

The maximum count of the alternate allele allowed in the normal sample.

--min-norm-depth [None]

The minimum depth required in the normal sample.

--min-tumor-alt-freq [None]

The minimum frequency of the alternate allele required in the tumor sample.

--min-tumor-alt-count [None]

The minimum count of the alternate allele required in the tumor sample.

--min-tumor-depth [None]

The minimum depth required in the tumor sample.

--chrom [None]

A specific chromosome on which to flag somatic mutations.

--dry-run

Don't set the is_somatic flag, just report what _would_ be set. For testing purposes.

actionable_mutations: Report actionable somatic mutations and drug-gene interactions

Actionable mutations are somatic variants in COSMIC cancer census genes with medium or high impact severity predictions. This tool reports actionable mutations as well as their known drug interactions (if any) from DGIdb. Current functionality is only for SNVs and indels.

Note:

1. **This tool requires somatic variants to have been flagged using** `set_somatic`

```
$ gemini actionable_mutations tumor_normal.db
tum_name        chrom   start   end     ref     alt     gene    impact  is_somatic
↪  in_cosmic_census        dgidb_info
tumor   chr2    128046288       128046289       C       T       ERCC3   non_syn_
↪coding  1       1       None
tumor   chr17   7578460 7578461 C       A       TP53    non_syn_coding  1       1
↪  {'searchTerm': 'TP53', 'geneCategories': ['CLINICALLY ACTIONABLE', 'DRUGGABLE␣
↪GENOME', 'TUMOR SUPPRESSOR', 'TRANSCRIPTION FACTOR COMPLEX', 'DRUG RESISTANCE',
↪'HISTONE MODIFICATION', 'DNA REPAIR', 'TRANSCRIPTION FACTOR BINDING'], 'geneName':
↪'TP53', 'geneLongName': 'tumor protein p53', 'interactions': [{'source': 'DrugBank',
↪ 'interactionId': '711cbe42-4930-4b46-963e-79ab35bbbd0f', 'interactionType': 'n/a',
↪'drugName': '1-(9-ETHYL-9H-CARBAZOL-3-YL)-N-METHYLMETHANAMINE'}, {'source':
↪'PharmGKB', 'interactionId': '8234d9b9-085d-49b1-aac2-cf5375d91477',
↪'interactionType': 'n/a', 'drugName': 'FLUOROURACIL'}, {'source': 'PharmGKB',
↪'interactionId': '605d7bca-7ed9-428e-aa7c-f76aafd66b54', 'interactionType': 'n/a',
↪'drugName': 'PACLITAXEL'}, {'source': 'TTD', 'interactionId': '1fe9db63-3581-435b-
↪b22a-12d45c8c9864', 'interactionType': 'activator', 'drugName': 'CURAXIN CBLC102'},
```

```
tumor   chr17   59861630        59861641        CTGCTATTTTG     CG      BRIP1
↪inframe_codon_loss     1       1       None
tumor   chrX    132838304       132838328       GAAAAAAAAAAAAAGGTGAAAATT
↪GAAAAAAAAAAAAGGTGAAAATT GPC3    splice_region   1       1       None
```

`fusions`: Report putative gene fusions

Report putative somatic gene fusions from structural variants in a tumor-normal pair. Putative fusions join two genes and preserve transcript strand orientation.

Note:

1. **This tool requires somatic variants to have been flagged using** `set_somatic`

default behavior

By default, `fusions` reports structural variants that are flagged as somatic, join two different genes, and preserve transcript strand orientation. These may be further filtered using tunable command line parameters.

```
    $ gemini fusions \
        --min_qual 5 \
        --in_cosmic_census \
        tumor_normal.db
    chromA   breakpointA_start  breakpointA_end     chromB  breakpointB_start
↪breakpointB_end var_id  qual    strandA strandB sv_type geneA   geneB   tool
↪evidence_type   is_precise  sample
chr3        176909953       176909982       chr3    178906001       178906030
↪1233    9.58    -       +       complex TBL1XR1 PIK3CA  LUMPY   PE      0
↪tumor
```

--min_qual [None]

The minimum required variant quality score.

--evidence_type STRING

The required supporting evidence types for the variant from LUMPY ("PE", "SR", or "PE,SR").

--in_cosmic_census

Require at least one of the affected genes to be in the COSMIC cancer gene census.

`db_info`: List the gemini database tables and columns

Because of the sheer number of annotations that are stored in gemini, there are admittedly too many columns to remember by rote. If you can't recall the name of particular column, just use the `db_info` tool. It will report all of the tables and all of the columns / types in each table:

```
$ gemini db_info test.db
table_name          column_name                 type
variants            chrom                       text
variants            start                       integer
variants            end                         integer
variants            variant_id                  integer
variants            anno_id                     integer
variants            ref                         text
variants            alt                         text
variants            qual                        float
variants            filter                      text
variants            type                        text
variants            sub_type                    text
variants            gts                         blob
variants            gt_types                    blob
variants            gt_phases                   blob
variants            gt_depths                   blob
variants            call_rate                   float
variants            in_dbsnp                    bool
variants            rs_ids                      text
variants            in_omim                     bool
variants            clin_sigs                   text
variants            cyto_band                   text
variants            rmsk                        text
variants            in_cpg_island               bool
variants            in_segdup                   bool
variants            is_conserved                bool
variants            num_hom_ref                 integer
variants            num_het                     integer
variants            num_hom_alt                 integer
variants            num_unknown                 integer
variants            aaf                         float
variants            hwe                         float
variants            inbreeding_coeff            float
variants            pi                          float
variants            recomb_rate                 float
variants            gene                        text
variants            transcript                  text
variants            is_exonic                   bool
variants            is_coding                   bool
variants            is_lof                      bool
variants            exon                        text
variants            codon_change                text
variants            aa_change                   text
variants            aa_length                   text
variants            biotype                     text
variants            impact                      text
variants            impact_severity             text
variants            polyphen_pred               text
variants            polyphen_score              float
variants            sift_pred                   text
variants            sift_score                  float
variants            anc_allele                  text
variants            rms_bq                      float
variants            cigar                       text
variants            depth                       integer
variants            strand_bias                 float
variants            rms_map_qual                float
```

```
variants            in_hom_run                  integer
variants            num_mapq_zero               integer
variants            num_alleles                 integer
variants            num_reads_w_dels            float
variants            haplotype_score             float
variants            qual_depth                  float
variants            allele_count                integer
variants            allele_bal                  float
variants            in_hm2                      bool
variants            in_hm3                      bool
variants            is_somatic
variants            in_esp                      bool
variants            aaf_esp_ea                  float
variants            aaf_esp_aa                  float
variants            aaf_esp_all                 float
variants            exome_chip                  bool
variants            in_1kg                      bool
variants            aaf_1kg_amr                 float
variants            aaf_1kg_asn                 float
variants            aaf_1kg_afr                 float
variants            aaf_1kg_eur                 float
variants            aaf_1kg_all                 float
variants            grc                         text
variants            gms_illumina                float
variants            gms_solid                   float
variants            gms_iontorrent              float
variants            encode_tfbs
variants            encode_consensus_gm12878    text
variants            encode_consensus_h1hesc     text
variants            encode_consensus_helas3     text
variants            encode_consensus_hepg2      text
variants            encode_consensus_huvec      text
variants            encode_consensus_k562       text
variants            encode_segway_gm12878       text
variants            encode_segway_h1hesc        text
variants            encode_segway_helas3        text
variants            encode_segway_hepg2         text
variants            encode_segway_huvec         text
variants            encode_segway_k562          text
variants            encode_chromhmm_gm12878     text
variants            encode_chromhmm_h1hesc      text
variants            encode_chromhmm_helas3      text
variants            encode_chromhmm_hepg2       text
variants            encode_chromhmm_huvec       text
variants            encode_chromhmm_k562        text
variant_impacts     variant_id                  integer
variant_impacts     anno_id                     integer
variant_impacts     gene                        text
variant_impacts     transcript                  text
variant_impacts     is_exonic                   bool
variant_impacts     is_coding                   bool
variant_impacts     is_lof                      bool
variant_impacts     exon                        text
variant_impacts     codon_change                text
variant_impacts     aa_change                   text
variant_impacts     aa_length                   text
variant_impacts     biotype                     text
variant_impacts     impact                      text
```

```
variant_impacts     impact_severity             text
variant_impacts     polyphen_pred               text
variant_impacts     polyphen_score              float
variant_impacts     sift_pred                   text
variant_impacts     sift_score                  float
samples             sample_id                   integer
samples             name                        text
samples             family_id                   integer
samples             paternal_id                 integer
samples             maternal_id                 integer
samples             sex                         text
samples             phenotype                   text
samples             ethnicity                   text
```

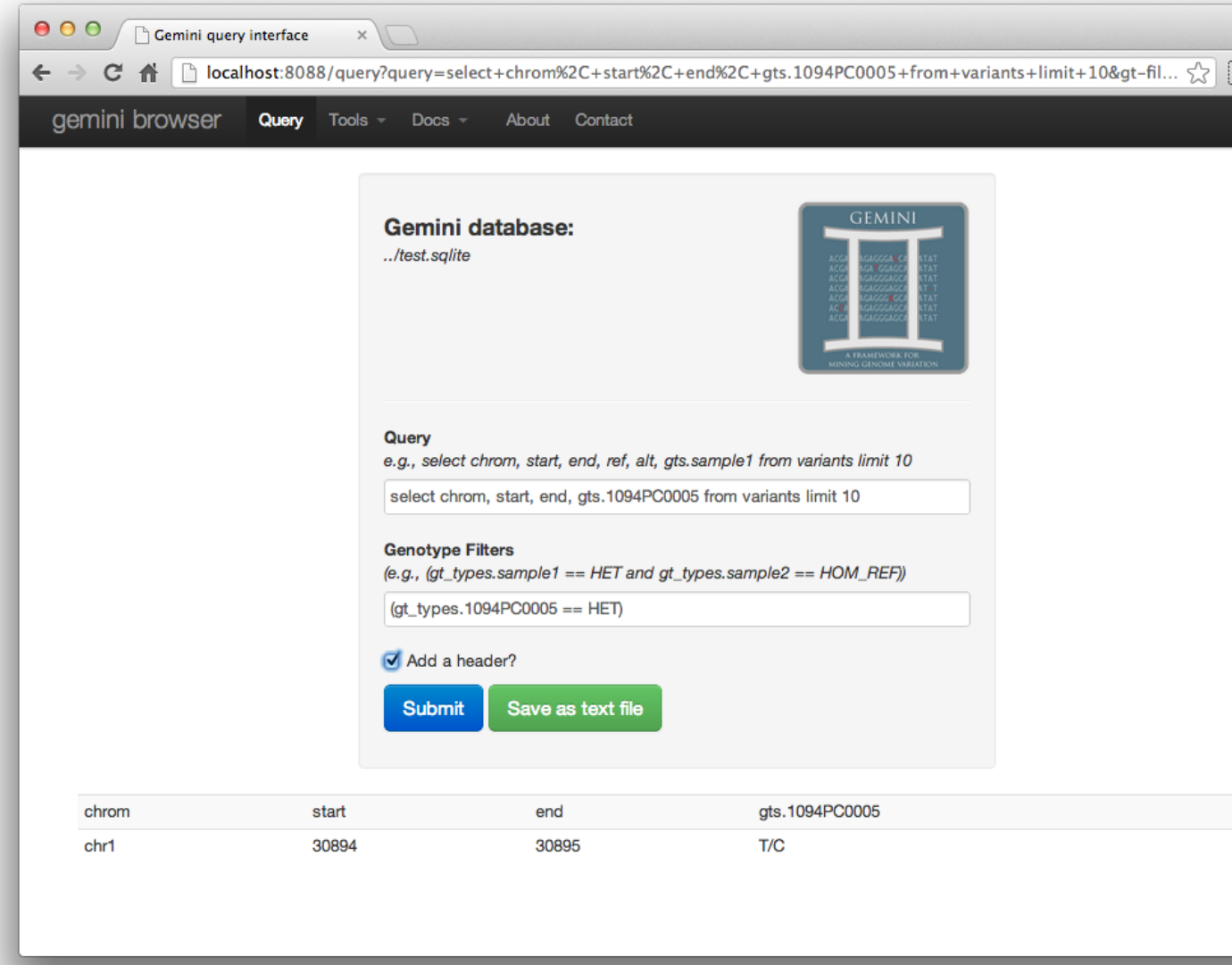
The GEMINI browser interface

Currently, the majority of GEMINI's functionality is available via a command-line interface. However, we are developing a browser-based interface for easier exploration of GEMINI databases created with the `gemini load` command.

Ironically, as of now, one must launch said browser from the command line as follows (where `my.db` should be replaced with the name of the GEMINI database you would like to explore).

```
$ gemini browser my.db --use builtin
```

At this point, the GEMINI browser is running on port 8088 on your local machine. Open a web browser to http://localhost:8088/query You should see something like:



NOTE: The internal (builtin) browser is in maintenace mode, please look at alternative browser support below for now.

Alternative browser support

GEMINI also allows to third_party genome browsers such as [puzzle](https://github.com/robinandeer/puzzle), the successor of [SciLifeLab](http://www.scilifelab.se/)'s [scout](https://github.com/robinandeer/scout).

If you want to spawn GEMINI-compatible browsers, you can use the following commandline:

```
$ gemini browser my.db [--use puzzle]
```

The GEMINI database schema

The `variants` table

Core VCF fields

column_name	type	notes
chrom	STRING	The chromosome on which the variant resides (from VCF `CHROM` field).
start	INTEGER	The 0-based start position. (from VCF `POS` field, but converted to 0-based coordinates)
end	INTEGER	The 1-based end position. (from VCF `POS` field, yet inferred based on the size of the variant)
vcf_id	STRING	The VCF `ID` field.
variant_id	INTEGER	PRIMARY_KEY
anno_id	INTEGER	Variant transcript number for the most severely affected transcript
ref	STRING	Reference allele (from VCF `REF` field)
alt	STRING	Alternate allele for the variant (from VCF `ALT` field)
qual	INTEGER	Quality score for the assertion made in ALT (from VCF `QUAL` field)
filter	STRING	A string of filters passed/failed in variant calling (from VCF `FILTER` field)

Variant and PopGen info

type	STRING	The type of variant. Any of: [*snp*, *indel*]
sub_type	STRING	The variant sub-type. If `type` is *snp*: [*ts*, (transition), *tv* (transversion)] If `type` is *indel*: [*ins*, (insertion), *del* (deletion)]
call_rate	FLOAT	The fraction of samples with a valid genotype
num_hom_ref	INTEGER	The total number of of homozygotes for the reference (`ref`) allele
num_het	INTEGER	The total number of heterozygotes observed.
num_hom_alt	INTEGER	The total number of homozygotes for the reference (`alt`) allele
num_unknown	INTEGER	The total number of of unknown genotypes
aaf	FLOAT	The observed allele frequency for the alternate allele
hwe	FLOAT	The Chi-square probability of deviation from HWE (assumes random mating)
inbreeding_coeff	FLOAT	The inbreeding co-efficient that expresses the likelihood of effects due to inbreeding
pi	FLOAT	The computed nucleotide diversity (pi) for the site

Genotype information

Gene information

gene	STRING	Corresponding gene name of the highly affected transcript
transcript	STRING	The variant transcript that was most severely affected (for two equally affected transcripts, the protein_coding biotype is prioritized (SnpEff/VEP)
is_exonic	BOOL	Does the variant affect an exon for >= 1 transcript?
is_coding	BOOL	Does the variant fall in a coding region (excl. 3' & 5' UTRs) for >= 1 transcript?
is_lof	BOOL	Based on the value of the impact col, is the variant LOF for >= transcript?
is_splicing	BOOL	Does the variant affect a canonical or possible splice site? That is, set to TRUE if the SO term is any of `splice_acceptor_variant`, `splice_donor_variant`, or `splice_region_variant`.
exon	STRING	Exon information for the severely affected transcript
codon_change	STRING	What is the codon change?
aa_change	STRING	What is the amino acid change (for a snp)?
aa_length	STRING	Has the format pos/len when biotype=protein_coding, is empty otherwise. len=protein length. pos = position of the amino acid change when is_coding=1 and is_exonic=1, '-' otherwise.
biotype	STRING	The 'type' of the severely affected transcript (e.g., protein-coding, pseudogene, rRNA etc.) (`only SnpEff`)
impact	STRING	The consequence of the most severely affected transcript
impact_so	STRING	The Sequence ontology term for the most severe consequence
impact_severity	STRING	Severity of the highest order observed for the variant
polyphen_pred	STRING	Polyphen predictions for the snps for the severely affected transcript (`only VEP`)
polyphen_score	FLOAT	Polyphen scores for the severely affected transcript (`only VEP`)
sift_pred	STRING	SIFT predictions for the snp's for the most severely affected transcript (`only VEP`)
sift_score	FLOAT	SIFT scores for the predictions (`only VEP`)
pfam_domain	STRING	Pfam protein domain that the variant affects

Optional VCF INFO fields

anc_allele	STRING	The reported ancestral allele if there is one.
rms_bq	FLOAT	The RMS base quality at this position.
cigar	STRING	CIGAR string describing how to align an alternate allele to the reference allele.
depth	INTEGER	The number of aligned sequence reads that led to this variant call
strand_bias	FLOAT	Strand bias at the variant position. From the "SB" tag.
rms_map_qual	FLOAT	RMS mapping quality, a measure of variance of quality scores
in_hom_run	INTEGER	Homopolymer runs for the variant allele
num_mapq_zero	INTEGER	Total counts of reads with mapping quality equal to zero
num_alleles	INTEGER	Total number of alleles in called genotypes
num_reads_w_dels	FLOAT	Fraction of reads with spanning deletions
haplotype_score	FLOAT	Consistency of the site with two segregating haplotypes
qual_depth	FLOAT	Variant confidence or quality by depth
allele_count	INTEGER	Allele counts in genotypes
allele_bal	FLOAT	Allele balance for hets
info	BLOB	Stores the `INFO` field of the VCF

Population information

in_dbsnp	BOOL	Is this variant found in dbSNP? 0 : Absence of the variant in dbsnp 1 : Presence of the variant in dbsnp
rs_ids	STRING	A comma-separated list of rs ids for variants present in dbSNP
in_hm2	BOOL	Whether the variant was part of HapMap2.
in_hm3	BOOL	Whether the variant was part of HapMap3.
in_esp	BOOL	Presence/absence of the variant in the ESP project data
in_1kg	BOOL	Presence/absence of the variant in the 1000 genome project data (phase 3)
aaf_esp_ea	FLOAT	Minor Allele Frequency of the variant for European Americans in the ESP project

Continued on next page

Table 5.3 – continued from previous page

aaf_esp_aa	FLOAT	Minor Allele Frequency of the variant for African Americans in the ESP project
aaf_esp_all	FLOAT	Minor Allele Frequency of the variant w.r.t both groups in the ESP project
aaf_1kg_amr	FLOAT	Allele frequency of the variant in AMR population based on AC/AN (1000g project, phase 3)
aaf_1kg_eas	FLOAT	Allele frequency of the variant in EAS population based on AC/AN (1000g project, phase 3)
aaf_1kg_sas	FLOAT	Allele frequency of the variant in SAS population based on AC/AN (1000g project, phase 3)
aaf_1kg_afr	FLOAT	Allele frequency of the variant in AFR population based on AC/AN (1000g project, phase 3)
aaf_1kg_eur	FLOAT	Allele frequency of the variant in EUR population based on AC/AN (1000g project, phase 3)
aaf_1kg_all	FLOAT	Global allele frequency (based on AC/AN) (1000g project - phase 3)
in_exac	BOOL	Presence/absence of the variant in ExAC (Exome Aggregation Consortium) data (Broad)
aaf_exac_all	FLOAT	Raw allele frequency (population independent) of the variant based on ExAC exomes (AF)
aaf_adj_exac_all	FLOAT	Adjusted allele frequency (population independent) of the variant based on ExAC (Adj_AC/Adj_AN)
aaf_adj_exac_afr	FLOAT	Adjusted allele frequency of the variant for AFR population in ExAC (AC_AFR/AN_AFR)
aaf_adj_exac_amr	FLOAT	Adjusted allele frequency of the variant for AMR population in ExAC (AC_AMR/AN_AMR)
aaf_adj_exac_eas	FLOAT	Adjusted allele frequency of the variant for EAS population in ExAC (AC_EAS/AN_EAS)
aaf_adj_exac_fin	FLOAT	Adjusted allele frequency of the variant for FIN population in ExAC (AC_FIN/AN_FIN)
aaf_adj_exac_nfe	FLOAT	Adjusted allele frequency of the variant for NFE population in ExAC (AC_NFE/AN_NFE)
aaf_adj_exac_oth	FLOAT	Adjusted allele frequency of the variant for OTH population in ExAC (AC_OTH/AN_OTH)

Continued on next page

Table 5.3 – continued from previous page

aaf_adj_exac_sas	FLOAT	Adjusted allele frequency of the variant for SAS population in ExAC (AC_SAS/AN_SAS)
max_aaf_all	FLOAT	the maximum of aaf_esp_ea, aaf_esp_aa, aaf_1kg_amr, aaf_1kg_eas,aaf_1kg_sas,aaf_1kg_afr,aaf_1kg_eur,aaf_ and -1 if none of those databases/populations contain the variant.
exac_num_het	INTEGER	The number of heterozygote genotypes observed in ExAC. Pulled from the ExAC `AC_Het` INFO field.
exac_num_hom_alt	INTEGER	The number of homozygous alt. genotypes observed in ExAC. Pulled from the ExAC `AC_Het` INFO field.
exac_num_chroms	INTEGER	The number of chromosomes underlying the ExAC variant call. Pulled from the ExAC `AN_Adj` INFO field.
aaf_gnomad_all	FLOAT	Allele frequency (population independent) of the variant in gnomad,
aaf_gnomad_afr	FLOAT	Allele frequency (AFR population) of the variant in gnomad
aaf_gnomad_amr	FLOAT	Allele frequency (AMR population) of the variant in gnomad
aaf_gnomad_asj	FLOAT	Allele frequency (ASJ population) of the variant in gnomad
aaf_gnomad_eas	FLOAT	Allele frequency (EAS population) of the variant in gnomad
aaf_gnomad_fin	FLOAT	Allele frequency (FIN population) of the variant in gnomad
aaf_gnomad_nfe	FLOAT	Allele frequency (NFE population) of the variant in gnomad
aaf_gnomad_oth	FLOAT	Allele frequency (OTH population) of the variant in gnomad
aaf_gnomad_sas	FLOAT	Allele frequency (SAS population) of the variant in gnomad
gnomad_num_het	INTEGER	Number of het genotypes observed in gnomad
gnomad_num_hom_alt	INTEGER	Number of hom_alt genotypes observed in gnomad
gnomad_num_chroms	INTEGER	Number of chromosomes genotyped in gnomad

Disease phenotype info (from ClinVar).

in_omim	BOOL	0 : Absence of the variant in OMIM database 1 : Presence of the variant in OMIM database
clinvar_causal_allele	STRING	The allele(s) that are associated or causal for the disease.
clinvar_sig	STRING	The clinical significance scores for each of the variant according to ClinVar: *unknown*, *untested*, *non-pathogenic* *probable-non-pathogenic*, *probable-pathogenic* *pathogenic*, *drug-response*, *histocompatibility* *other*
clinvar_disease_name	STRING	The name of the disease to which the variant is relevant
clinvar_dbsource	STRING	Variant Clinical Channel IDs
clinvar_dbsource_id	STRING	The record id in the above database
clinvar_origin	STRING	The type of variant. Any of: *unknown*, *germline*, *somatic*, *inherited*, *paternal*, *maternal*, *de-novo*, *biparental*, *uniparental*, *not-tested*, *tested-inconclusive*, *other*
clinvar_dsdb	STRING	Variant disease database name
clinvar_dsdbid	STRING	Variant disease database ID
clinvar_disease_acc	STRING	Variant Accession and Versions
clinvar_in_locus_spec_db	BOOL	Submitted from a locus-specific database?
clinvar_on_diag_assay	BOOL	Variation is interrogated in a clinical diagnostic assay?
clinvar_gene_phenotype	STRING	'l' delimited list of phenotypes associated with this gene (includes any variant in the same gene in clinvar not just the current variant).
geno2mp_hpo_ct	INTEGER	Value from geno2mp indicating count of HPO profiles. Set to -1 if missing

Structural variation columns

sv_cipos_start_left	INTEGER	The leftmost position of the leftmost SV breakpoint confidence interval.
sv_cipos_end_left	INTEGER	The rightmost position of the leftmost SV breakpoint confidence interval.
sv_cipos_start_right	INTEGER	The leftmost position of the rightmost SV breakpoint confidence interval.
sv_cipos_end_right	INTEGER	The rightmost position of the rightmost SV breakpoint confidence interval.
sv_length	INTEGER	The length of the structural variant in base pairs.
sv_is_precise	BOOL	Is the structural variant precise (i.e., to 1-bp resolution)?
sv_tool	STRING	The name of the SV discovery tool used to find the SV.
sv_evidence_type	STRING	What type of alignment evidence supports the SV?
sv_event_id	STRING	A unique identifier for the SV.
sv_mate_id	STRING	The ID for the “other end” of the SV.
sv_strand	STRING	The orientations of the SV breakpoint(s).

Genome annotations

exome_chip	BOOL	Whether a SNP is on the Illumina HumanExome Chip
cyto_band	STRING	Chromosomal cytobands that a variant overlaps
rmsk	STRING	A comma-separated list of RepeatMasker annotations that the variant overlaps. Each hit is of the form: `name_class_family`
in_cpg_island	BOOL	Does the variant overlap a CpG island?. Based on UCSC: Regulation > CpG Islands > cpgIslandExt
in_segdup	BOOL	Does the variant overlap a segmental duplication?. Based on UCSC: Variation&Repeats > Segmental Dups > genomicSuperDups track
is_conserved	BOOL	Does the variant overlap a conserved region? Based on the 29-way mammalian conservation study
gerp_bp_score	FLOAT	GERP conservation score. Only populated if the `--load-gerp-bp` option is used when loading. Higher scores reflect greater conservation. **At base-pair resolution**. Details: http://mendel.stanford.edu/SidowLab/downloads/gerp/
gerp_element_pval	FLOAT	GERP elements P-val Lower P-values scores reflect greater conservation. **Not at base-pair resolution**. Details: http://mendel.stanford.edu/SidowLab/downloads/gerp/
recomb_rate	FLOAT	Returns the mean recombination

Note: CADD scores (http://cadd.gs.washington.edu/) are Copyright 2013 University of Washington and Hudson-Alpha Institute for Biotechnology (all rights reserved) but are freely available for all academic, non-commercial applications. For commercial licensing information contact Jennifer McCullar (mccullaj@uw.edu).

Variant error assessment

grc	STRING	Association with patch and fix regions from the Genome Reference Consortium: http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/ Identifies potential problem regions associated with variant calls. Built with *annotation_provenance/make-ncbi-grc-patches.py*
gms_illumina	FLOAT	Genome Mappability Scores (GMS) for Illumina error models Provides low GMS scores (< 25.0 in any technology) from: http://sourceforge.net/apps/mediawiki/gma-bio/index.php?title=Download_GMS #Download_GMS_by_Chromosome_and_Sequencing_ Input VCF for annotations prepared with: https://github.com/chapmanb/bcbio.variation/blob/master/src/bcbio/variation/utils/gms.clj
gms_solid	FLOAT	Genome Mappability Scores with SOLiD error models
gms_iontorrent	FLOAT	Genome Mappability Scores with IonTorrent error models
in_cse	BOOL	Is a variant in an error prone genomic position, using CSE: Context-Specific Sequencing Errors https://code.google.com/p/discovering-cse/ http://www.biomedcentral.com/1471-2105/14/S5/S1

ENCODE information

encode_tfbs	STRING	Comma-separated list of transcription factors that were observed by ENCODE to bind DNA in this region. Each hit in the list is constructed as TF_CELLCOUNT, where: *TF* is the transcription factor name *CELLCOUNT* is the number of cells tested that had nonzero signals. Provenance: wgEncodeRegTfbsClusteredV2 UCSC table
encode_dnaseI_cell_count	INTEGER	Count of cell types that were observed to have DnaseI hypersensitivity.
encode_dnaseI_cell_list	STRING	Comma separated list of cell types that were observed to have DnaseI hypersensitivity. Provenance: Thurman, et al, *Nature*, 489, pp. 75-82, 5 Sep. 2012
encode_consensus_gm12878	STRING	ENCODE consensus segmentation prediction for GM12878. CTCF: CTCF-enriched element E: Predicted enhancer PF: Predicted promoter flanking region R: Predicted repressed or low-activity region TSS: Predicted promoter region including TSS T: Predicted transcribed region WE: Predicted weak enhancer or open chromatin cis-regulatory element | unknown: This region of the genome had no functional prediction.
encode_consensus_h1hesc	STRING	ENCODE consensus segmentation prediction for h1hesc. See encode_consseg_gm12878 for details.
encode_consensus_helas3	STRING	ENCODE consensus segmentation prediction for Helas3. See encode_consseg_gm12878 for details.

Cancer related columns

is_somatic	BOOL	Whether the variant is somatically acquired.
cosmic_ids	STRING	A list of known COSMIC ids for this variant. See: http://cancer.sanger.ac.uk/cancergenome/projects/cosmic/

The `variant_impacts` table

column_name	type	notes
variant_id	INTEGER	PRIMARY_KEY (Foreign key to *variants* table)
anno_id	INTEGER	PRIMARY_KEY (Based on variant transcripts)
gene	STRING	The gene affected by the variant.
transcript	STRING	The transcript affected by the variant.
is_exonic	BOOL	Does the variant affect an exon for this transcript?
is_coding	BOOL	Does the variant fall in a coding region (excludes 3' & 5' UTR's of exons)?
is_lof	BOOL	Based on the value of the impact col, is the variant LOF?
exon	STRING	Exon information for the variants that are exonic
codon_change	STRING	What is the codon change?
aa_change	STRING	What is the amino acid change?
aa_length	STRING	The length of CDS in terms of number of amino acids (`SnpEff only`)
biotype	STRING	The type of transcript (e.g., protein-coding, pseudogene, rRNA etc.) (`SnpEff only`)
impact	STRING	Impacts due to variation (ref.impact category)
impact_so	STRING	The sequence ontology term for the impact
impact_severity	STRING	Severity of the impact based on the impact column value (ref.impact category)
polyphen_pred	STRING	Impact of the SNP as given by PolyPhen (`VEP only`) benign, possibly_damaging, probably_damaging, unknown
polyphen_scores	FLOAT	Polyphen score reflecting severity (higher the impact, *higher* the score) (`VEP only`)
sift_pred	STRING	Impact of the SNP as given by SIFT (`VEP only`) neutral, deleterious
sift_scores	FLOAT	SIFT prob. scores reflecting severity (Higher the impact, *lower* the score) (`VEP only`)

Details of the `impact` and `impact_severity` columns

impact severity	impacts	SO_impacts
HIGH	• exon_deleted • frame_shift • splice_acceptor • splice_donor • start_loss • stop_gain • stop_loss • non_synonymous_start • transcript_codon_change • rare_amino_acid • chrom_large_del	• exon_loss_variant • frameshift_variant • splice_acceptor_variant • splice_donor_variant • start_lost • stop_gained • stop_lost • initiator_codon_variant • initiator_codon_variant • rare_amino_acid_variant • chromosomal_deletion
MED	• non_syn_coding • inframe_codon_gain • inframe_codon_loss • inframe_codon_change • codon_change_del • codon_change_ins • UTR_5_del • UTR_3_del • splice_region • mature_miRNA • regulatory_region • TF_binding_site • regulatory_region_ablation • regulatory_region_amplification • TFBS_ablation • TFBS_amplification	• missense_variant • inframe_insertion • inframe_deletion • coding_sequence_variant • disruptive_inframe_deletion • disruptive_inframe_insertion • 5_prime_UTR_truncation + exon_loss_variant • 3_prime_UTR_truncation + exon_loss_variant • splice_region_variant • mature_miRNA_variant • regulatory_region_variant • TF_binding_site_variant • regulatory_region_ablation • regulatory_region_amplification • TFBS_ablation • TFBS_amplification
LOW	• synonymous_stop • synonymous_coding • UTR_5_prime • UTR_3_prime • intron • CDS • upstream • downstream • intergenic • intragenic • gene • transcript • exon • start_gain • synonymous_start • intron_conserved • nc_transcript • NMD_transcript • incomplete_terminal_codon • nc_exon • transcript_ablation • transcript_amplification • feature elongation	• stop_retained_variant • synonymous_variant • 5_prime_UTR_variant • 3_prime_UTR_variant • intron_variant • coding_sequence_variant • upstream_gene_variant • downstream_gene_variant • intergenic_variant • intragenic_variant • gene_variant • transcript_variant • exon_variant • 5_prime_UTR_premature_start_codon_gain_vari • start_retained_variant • conserved_intron_variant • nc_transcript_variant • NMD_transcript_variant • incomplete_terminal_codon_variant • non_coding_exon_variant • transcript_ablation

The `samples` table

column name	type	notes
sample_id	INTEGER	PRIMARY_KEY
name	STRING	Sample names
family_id	INTEGER	Family ids for the samples [User defined, default: NULL]
paternal_id	INTEGER	Paternal id for the samples [User defined, default: NULL]
maternal_id	INTEGER	Maternal id for the samples [User defined, default: NULL]
sex	STRING	Sex of the sample [User defined, default: NULL]
phenotype	STRING	The associated sample phenotype [User defined, default: NULL]
ethnicity	STRING	The ethnic group to which the sample belongs [User defined, default: NULL]

The `resources` table

Establishes provenance of annotation resources used to create a GEMINI database.

column name	type	notes
name	STRING	Name of the annotation type
resource	STRING	Filename of the resource, with version information

The `version` table

Establishes which version of `gemini` was used to create a database.

column name	type	notes
version	STRING	What version of gemini was used to create the DB.

The `gene_detailed` table

Built on version 75 of Ensembl genes

column_name	type	notes
uid	INTEGER	PRIMARY_KEY (unique identifier for each entry in the table)
chrom	STRING	The chromosome on which the gene resides
gene	STRING	The gene name
is_hgnc	BOOL	Flag for gene column: 0 for non HGNC symbol and 1 for HGNC symbol = TRUE
ensembl_gene_id	STRING	The ensembl gene id for the gene
transcript	STRING	The ensembl transcript id for the gene
biotype	STRING	The biotype (e.g., protein coding) of the transcript
transcript_status	STRING	The status of the transcript (e.g. KNOWN, PUTATIVE etc.)
ccds_id	STRING	The consensus coding sequence transcript identifier
hgnc_id	STRING	The HGNC identifier for the gene if HGNC symbol is TRUE
entrez_id	STRING	The entrez gene identifier for the gene
cds_length	STRING	The length of CDS in bases
protein_length	STRING	The length of the transcript as the number of amino acids
transcript_start	STRING	The start position of the transcript in bases
transcript_end	STRING	The end position of the transcript in bases
strand	STRING	The strand of DNA where the gene resides
synonym	STRING	Other gene names (previous or synonyms) for the gene
rvis_pct	FLOAT	The RVIS percentile values for the gene
mam_phenotype_id	STRING	High level mammalian phenotype ID applied to mouse phenotype descriptions in the MGI database at http://www.informatics.jax.org/. Data taken from ftp://ftp.informatics.jax.org/pub/reports/HMD_HumanPhenotype.rpt

The `gene_summary` table

Built on version 75 of Ensembl genes

column_name	type	notes
uid	INTEGER	PRIMARY_KEY (unique identifier for each entry in the table)
chrom	STRING	The chromosome on which the gene resides
gene	STRING	The gene name
is_hgnc	BOOL	Flag for gene column: 0 for non HGNC symbol and 1 for HGNC symbol = TRUE
ensembl_gene_id	STRING	The ensembl gene id for the gene
hgnc_id	STRING	The HGNC identifier for the gene if HGNC symbol is TRUE
transcript_min_start	STRING	The minimum start position of all transcripts for the gene
transcript_max_end	STRING	The maximum end position of all transcripts for the gene
strand	STRING	The strand of DNA where the gene resides
synonym	STRING	Other gene names (previous or synonyms) for the gene
rvis_pct	FLOAT	The RVIS percentile values for the gene
mam_phenotype_id	STRING	High level mammalian phenotype ID applied to mouse phenotype descriptions in the MGI database at http://www.informatics.jax.org/. Data taken from ftp://ftp.informatics.jax.org/pub/reports/HMD_HumanPhenotype.rpt
in_cosmic_census	BOOL	Are mutations in the gene implicated in cancer by the cancer gene census?

Using the GEMINI API

The GeminiQuery class

Extracting the VCF INFO tags with GEMINI API

The GEMINI API is useful to extract the individual tags within the INFO field of a VCF (stored as a compressed dictionary in the variants table). This would be of particular interest to those who want to add custom annotations to their VCF and still be able to access the individual tags programmatically. Here is an example where we try to extract the dbNSFP fields from the 'INFO' tag of a VCF, using the API.

```
#!/usr/bin/env python
import sys
from gemini import GeminiQuery
```

```
database = sys.argv[1]
gq = GeminiQuery(database)
query = "SELECT variant_id, chrom, start, end, ref, alt, info \
         FROM variants"

gq.run(query)

for row in gq:
    try:
        print "\t".join([str(row['chrom']), str(row['start']), str(row['end']),
                         str(row['ref']), str(row['alt']), str(row.info['dbNSFP_SIFT_pred
↪'])])
    except KeyError:
        pass

# yields
chr1    906272  906273  C       T       P|D|P
chr1    906273  906274  C       A       D|D|D
chr1    906276  906277  T       C       D|D|D
chr1    906297  906298  G       T       B|B|B
chr1    1959074 1959075 A       C       D
chr1    1959698 1959699 G       A       B
chr1    1961452 1961453 C       T       P
chr1    2337953 2337954 C       T       D
```

Speeding genotype queries

Design of Gemini

Gemini stores all genotypes information in a single column. E.g. *gt_depths* is a single column in the sqlite database that contains information on all samples in a compressed and serialized array. So, in order to do a query involving *–gt-filter gemini* must iterate over each row, decompress and de-serialize the array and then evaluate the genotype filter. Even if the filter involves only a single sample, we must deserialize the entire array. This design works quite well and we have improved performance greatly, but for complex queries, it is quite slow.

We provide the means to index genotype fields and *optionally* use those indexes to perform the genotype filtering. These are external to gemini in that they do not change the behavior of *gemini* when used without the engine.

bcolz indexes

We have a implemented indexes using bcolz.

These can be used for existing databases by creating the external indexes:

```
gemini bcolz_index $db
```

This is easily parallelized by specifying a column per process, e.g.

```
gemini bcolz_index $db --cols gt_types
gemini bcolz_index $db --cols gt_depths # in another process
```

Which can index about 20K variants / second for 17 samples in our testing. Due to a limitation in the string type in bcolz, indexing the *gts* column is slower so we recommend not doing those queries with bcolz (e.g. gts.sample1 == 'A/C'). See note below.

It is recommended to only index the columns you'll be using in the *–gt-filter*.

Indexing is done only once; after that, add *–use-bcolz* to an existing gemini query command. e.g.

```
gemini query -q "select variant_id, gts.1719PC0016 from variants"  \
    --gt-filter "gt_types.1094PC0012 == HET and gt_types.1719PC0016 == HET and gts.
↪1094PC0012 == 'A/C'" \
    --use-bcolz \
    test/test.query.db
```

This query will return identical results with or without using bcolz. It is likely only beneficial to use *bcolz* on complex queries that are slow with the default gemini apparatus. Queries that are more selective–e.g. return fewer rows are the best target for speed improvement with *bcolz*

An example of the types of improvements (usually 20X to 50X) with various queries is here.

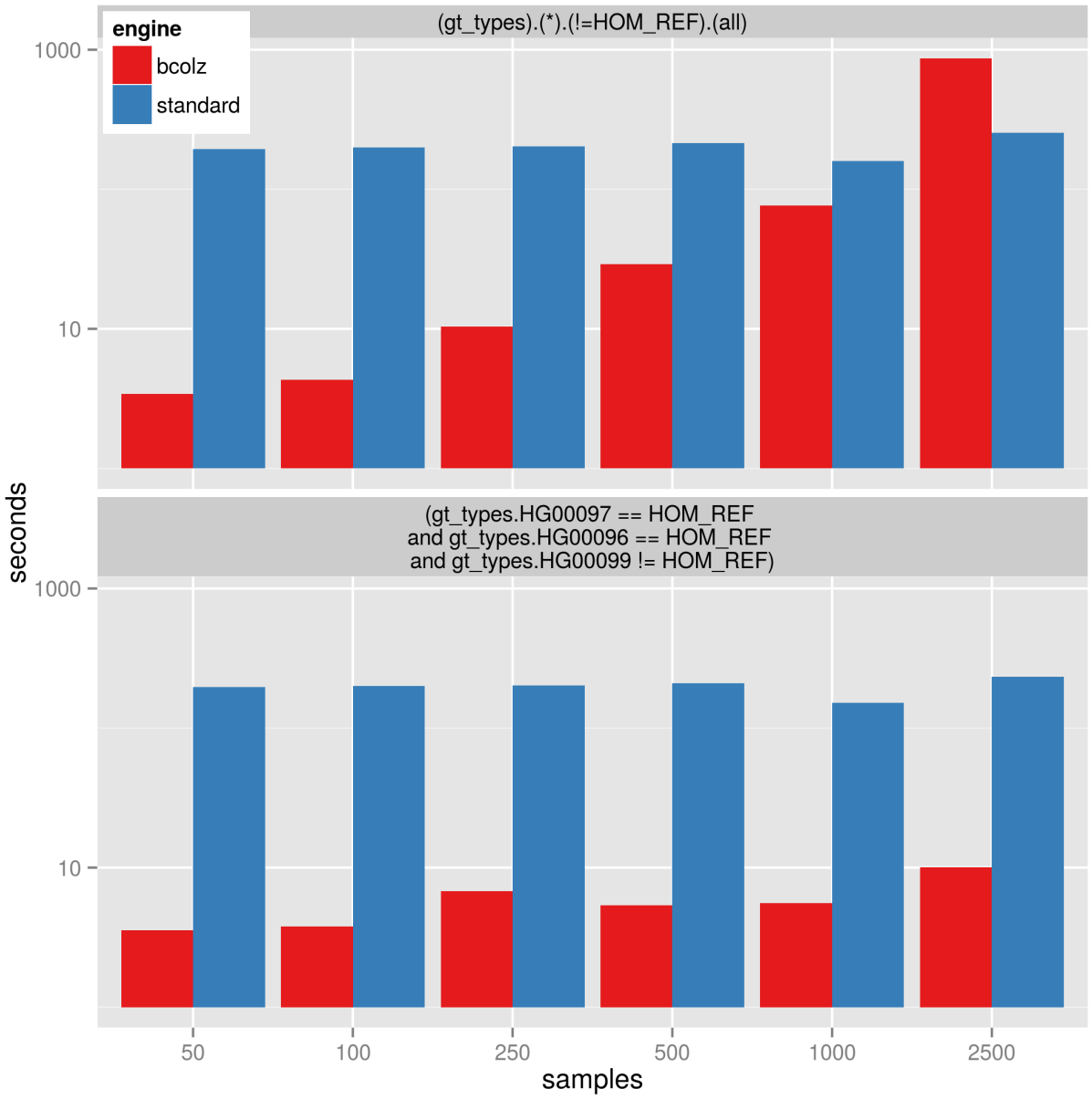
limitations

As the number of samples grows, it becomes less beneficial to use *–gt-filter* s that touch all samples. For example: *(gt_types).(*).(!=HOM_REF).(all)* will become slower as samples are added since it must test every sample. However, a query like: *(gt_types.DAD == HOM_REF and gt_types.MOM == HOM_REF and gt_types.KID != HOM_REF)* will be much less affected by the number of samples in the database because they are only touching 3 samples.

The image below shows the time to perform the filters on a database with 1.8 million variants and varying sample size:

1. *(gt_types).(*).(!=HOM_REF).(all)* which is affected by sample size
2. *(gt_types.DAD == HOM_REF and gt_types.MOM == HOM_REF and gt_types.KID != HOM_REF)* which is less affected by sample-size

Note that at 2500 samples, using bcolz is slower than the standard gemini query, however using bcolz is consistently 30 to 50 times faster for the 2nd query. (This is up to 1000 times faster than versions of gemini before 0.12). The y-axis is log10-scaled.



Note: indexing the 'gts' column will be much slower (only about 350 variants per second instead of up to 25K per second) as it must be stored as an object rather than a fixed-size numeric type. It will be a much larger index. So only create an index on 'gts' if necessary.

Usage

First, make sure you have gemini version 0.15 or greater:

```
gemini --version
```

Then, you can index an existing database (*$db*) with:

```
gemini bcolz_index $db
```

Then, wherever you have a slow query that's using the *–gt-filter*, you can add *–use-bcolz* to the query command.

```
gemini query -q "select chrom, start, end from variants" $db \
        --use-bcolz \
        --gt-filter "gt_depth.samples1 >= 20 and gt_depth.samples2 >= 20 and gt_depth.
↪samples3 >= 20 \
          and (gt_types.sample1 == HOM_REF and gt_types.sample2 == HOM_REF and gt_
↪types.sample3 != HOM_REF)"
```

Note that nothing has changed except that *–use-bcolz* is added to the query.

Design of Genotype Query Engines

This sections is for those wishing to create their own genotype query engines to plug in to gemini and will not be needed for most users.

Genotype Query Engines can be plugged in to *gemini*. They must be exposed with a single function:

> filter(db_path, gt_filter, user_dict)

where *db_path* is the path to the gemini sqlite database, *gt_filter* is the genotype query string. user_dict will be pre-filled with things like user_dict contains things like HET, UNKNOWN, etc. used in gemini.

The *filter* function must return a list of integer variant_ids that meet the specified filter. If it can not perform the query, it must return *None*.

gemini will internally use the returned variant_ids to modifiy the sqlite query to select only those rows.

The *filter* function only needs to worry about which variant_ids to return, not how to integrate with the rest of *gemini*.

Acknowledgements

GEMINI is developed by Uma Paila and Aaron Quinlan in the Quinlan laboratory at the University of Virginia. Substantial contributions to the design, functionality, and code base have been made by the following:

- Brad Chapman, HSPH
- Rory Kirchner, HSPH
- Oliver Hofmann, HSPH
- Björn Grüning, University of Freiburg

Release History

0.21.0 (future)

1. Use vcfanno for faster, more generalized variant annotation and database creation.

0.20.1

1. Allow *–gt-filter* to specify the gt_type, e.g. *(gt_quals).(=HET).(<20).(all)* (thanks Jessica for suggesting)
2. Add *–gene-where* argument to *comp_hets* tool to dictate how variants are restricted to genes. (thanks Jessica for suggesting)
3. genewise: don't show any variants that don't meet the required number of optional filters.
4. Fix AF for decomposed gnomad variants (thanks @mbootwalla).

0.20.0

1. Include gnomad exome allele frequencies for all populations and include these in calc of max_aaf_all.
2. Initial support for python 3.
3. Support annotating with bcftools BCSQ (http://biorxiv.org/content/early/2016/12/01/090811) in addition to VEP and SnpEff.
4. Fix bug that resulted in the message: *UnsupportedOperation: IOStream has no fileno.* when loading. (Thanks Andrew O. and Ryan R.)
5. Fix bug where gemini annotate would never finish (see: arq5x/gemini#809 (Thanks @mmoisse)
6. Add extra columns from VEP when -t all is used (previously extra VEP columns were ignored under -t all. thanks @hoppman #816).
7. Fix gemini annotate (would stop after updating 100K variants)
8. Fix browser query report (thanks @Nmael see #818).
9. Allow specifying a –where clause for *gene_wise* to filter considered variants. (Thanks Uma for suggestion).
10. Fix bug in propagating vep-extras to variant_impacts table (Thanks Sergey for reporting).
11. Update clinvar annotation.
12. Add new *gemini load* flag *–skip-pls* which can give size reductions for the database and speed Improvements for querying and loading. If you never use *gt_phred_ll_homalt/het/homref* then you can load with this flag.
13. Add new *gt_alt_freqs* column which is *gt_alt_dephts / (gt_ref_depths + gt_alt_depths)*.
14. Update CADD to v1.3

0.19.1

1. Fix bug in `comp_het` where candidates were not removed when either parent was homozygous at both sites (thanks Jessica Chong). (inheritance v0.1.0)
2. Update `x-linked-dominant` with the following rules (thanks Jessica Chong): + mothers of affected males must be het (and affected) + at least 1 parent of affected females must be het (and affected).
3. Store extra vep fields for each transcript in variant_impacts.
4. Update geneimpacts module to handle new snpEff annotations of SVs (Brad Chapman).
5. Update to dbsnp 147.

0.19.0

1. Use SQLAlchemy for table definitions to support different RDBMS back-ends.
2. Several optimizations to loading.
3. X-linked recessive, dominant, and de novo tools.
4. Raise exceptions rather than sys.exit() to facilitate use as library. (thanks @brainstorm).
5. The `comp_het` tool is as much as 20X faster for large cohorts.

0.18.3

1. handle rare VEP annotation with '' or "?" as impact
2. fix handling of multiple values for list/uniq_list in gemini_annotate
3. handle snpEff with unknown impact (TODO: bump geneimpacts req to 0.1.0)
4. fix builtin browser and add support for puzzle browser
5. add –gt-filter-required to gemini gene_wise tool that must pass for each variant.
6. fix bug in lof_sieve when transcript pos is not specified. (thanks @mmoisse)

0.18.2

1. Update the Clinvar annotation file to the February 3, 2016 release.

0.18.1

1. Add `clinvar_gene_phenotype` column which can be used to limit candidates to those that are in the same gene as a gene with the known phenotype from clinvar. Phenotypes are all lower case. Likely usage is: `--filter "... and clinvar_gene_phenotype LIKE '%dysplasia%'` where the '%' wildcard is needed because the column is a 'l' delimited list of all disease for that gene.
2. Add `geno2mp_hpo_ct` column which will be > 0 if that variant is present in geno2mp (http://geno2mp.gs.washington.edu/Geno2MP/#/)
3. Fix `comp_hets` tool when `exac_num_hom_alt` was requested. (Thanks to @davemcg for reporting).
4. Change `splice_region_variant` from `LOW` to `MED` priority in the interest of reducing false negatives in the study of rare disease.
5. Set missing AF to -1 (instead of NULL) for unknown for all ESP, 1KG, ExAC allele frequency columns.
6. Don't user `order by` when not needed for built in tools. Speeds up queries when min-kindreds is None or 1.
7. Document the `--min-gq` option
8. Fix gemini load with `-t all` when only VEP is present.
9. Fix bug in gemini `annotate` where numeric operations on integers did not work for VCF.
10. Fix bug in comp_hets tool where attempting to phase a "." genotype would result in an error. (Thanks to Aparna and Martina for reporting)
11. Fix bug in `mendel_errors tool` when specifying `--families`. Thanks to @davemcg for reporting.

0.18.0

0. Improved installation and update via conda (via Brad Chapman!).
1. Update support for SO term variant impact predictions via VEP and SnpEff and support newest snpEff version. If both snpEff and VEP annotations are present use *gemini load -t all* to save all annotations and the most deleterious will be saved in the variants table.
2. Add an *is_splicing* column.
3. Add *exac_num_het*, *exac_num_hom_alt* and *exac_num_chroms* columns. (See #568).
4. Fix issues with cyvcf2 handling of haploid calls (from X chromosome from GATK; thanks Athina for reporting)
5. Fix handling of VEP extra fields (thanks @jsh58).
6. Remove pygraph dependency (networkx is easier to install). Allow specifying custom edges to interactions tools.
7. Fix some edge-cases in compound het tool. Thanks to Jamie Kwok for reporting.
8. Allow requring a minimum genotype quality in inheritance tools with –min-gq option.
9. Fix bug in cyvcf2 for parsing hemizyous (0/.) variants from decomposition.
10. *comp_het* tool now has a priority 2 level for unphased singletons where both sites are HET. Candidates where both parents and HET, along with the affected are now priority 3.

0.17.2

1. Fix bcolz dependencies owing to issues with bcolz 0.11.0

0.17.1

1. Change handling of missing values in PL/GL (–gt-pl-max) so that missing values are set to int32 max. (Thanks Karyn for reporting).
2. Fix distributed loading of VEP with extra columns (@chapmanb) [Regression since 0.17.0]
3. Fix comp_het test and improve efficiency (thanks Bianca for reporting)
4. Bug fix: populate eval dictionary with sample_info.

0.17.0

1. switch to cyvcf2 to speed loading
2. per-sample depths are calculated using AD (GATK) or AO+RO (Freebayes). This makes depth filters more conservative.
3. extra VEP annotations are loaded with loading machinery, not as an extra step as before.
4. add max_aaf_all column (https://github.com/arq5x/gemini/issues/520) as an aggregate of a number of population filters.
5. use –families to limit queries *before* any work is done. Thanks to Bianca for reporting.
6. No longer create bcolz indicies by default. Users can create them with *gemini bcolz_index*.
7. New *genewise* tool. See docs.

8. gemini load: –skip-info-string has been replaced with –save-info-string and the INFO field is not longer saved by default.
9. comp_hets: default to only showing confident (priority 1) candidates. Show all candidates with –max-priority 3.

0.16.3

1. Fix bug in `comp_het` with reporting same pair multiple times.
2. Handle UNKOWN genotypes in `comp_het` tool
3. Fix cyvcf dependency in requirements
4. Only run tests that require bgzip/tabix/bedtools if they are available on PATH
5. Limit ipython version to 3<version<4

0.16.2

1. Hone rules for unphased and partially-phased compound hets.
2. Remove *–lenient* argument for comp_hets and add *–pattern-only* to find compound_hets regardless of affection status.
3. The *–lenient* argument to the *autosomal_dominant* tool has been relaxed to allow parents with unknown phenotypes.
4. Re (vt) decompose data files for 1000 genomes and ExAC (thanks Julien and Xiaolin for reporting).

0.16.1

1. Fix regression in loading when AAF is None
2. Fix handing in mendelian error tool where all genotype likelihoods are low (thanks Bianca)
3. Don't phase de-novo's (caused error in comp_het tool). (thanks Bianca)
4. Fix regression in loading VEP with multicore (thanks Andrew)

0.16.0

1. The built-in inheritance model tools (`auto_rec`, etc.) have been modified to be more restrictive in order to remove false positive candidates. The strictness can be reduced by using the `--lenient` option.
2. Leverage bcolz indexing for the built-in inheritance model tools to dramatically improve speed.
3. Support for multi-generational pedigrees for the built in inheritance model tools. (thanks to Jessica, Andrew, and jmcelwee for extensive discussion https://github.com/arq5x/gemini/issues/388)
4. Leverage genotype likelihoods in tools other than `mendel_errors` as a means to filter variants.
5. Automatically phase genotypes by transmission on the fly for the *comp_hets* tool.
6. Further performance improvements for bcolz queries
7. The `--affected-only` option has been made the default and it's opposing replacement named `--allow-unaffected` to revert.

8. Fixed a reporting error for the inheritance tools (i.e., family_id was mis-specified in output).
9. Annotate the variants table with impact even if there is not severe impact. Thanks to @mjsduncan for reporting.
10. Reduce memory requirements when loading. Thanks to @mjsduncan for reporting.

0.15.1

1. Fix regression in grabix. Thanks to Sven-Eric Shelhorn for reporting.
2. Fix handling of samples with "-". Thanks to Uma Paila for reporting.

0.15.0

1. Use external index to speed genotype queries (this is created by default on load unless –no-bcolz is specified)
2. Match on ref and alternate alleles (not just position) when annotating with VCF. Thanks Jeremy Goecks.
3. Related to matching, we now load extra annotation, e.g. VEP as VCF and require ref and alt matching. Previously was done with bed overlap.
4. Faster queries due to lazy loading of genotype columns.
5. Read gt* columns from the database for better backward compatibility.
6. Code cleanup. Thanks to Christian Brueffer.

0.14.0

1. Standardized the output from the built-in tools into a common, BED+ format. Thanks to feedback from Jessica Chong and Daniel Gaston.
2. Release of *mendel_errors* tool which also outputs the type of error and the probability (based on PL's)
3. Improvements to the *load* tool when running on large compute clusters using PBS, SGE, SLURM, etc. Also provde a workaround for NFS locking issues. Many thanks to Ben Weisburd in Daniel Macarthur's lab.
4. Improve preprocess script to support varscan, platypus (https://gist.github.com/brentp/4db670df147cbd5a2b32)
5. Performance improvements for many of the built-in tools (pre-compile evals)
6. Bug fix for installation with sudo privileges.

0.13.1 (2015-Apr-09)

1. Major *query* speed improvements. For example, the following query goes from 43 seconds in version 0.12.2 to 11 seconds in 0.13.0. All queries involving *gt_** fields should be substantially faster.

```
$ gemini query \
        -q "select chrom, start, (gts).(*) from variants" data/tmaster.db \
        --gt-filter "(gt_depths).(*).(>=20).(all)" > /dev/null
```

2. Speed improvements to *load*. The following went from 7 minutes 9 seconds to 6 minutes 21 seconds.

```
$ gemini load -t VEP -v data/v100K.vcf.gz data/tmaster.db --cores 4
```

3. We added the *gt_phred_ll_homref*, *gt_phred_ll_het*, *gt_phred_ll_homalt* columns to database. These are the genotype likelihoods pulled from the GL or PL columns of the VCF if available. They can all be queried and filtered in the same way as existing gt_* columns. In future releases, we are planning tp use genotype likelihood to assign likelihoods to de novo mutations, mendelian violations, and variants meeting other inheritance patterns.
4. Fixed bugs related to splitting multiple alts (thanks to @jdh237)
5. We are working to improve development and release testing. This is ongoing, but we now support gemini_install.py –version unstable so that users can try out the latest changes and help with testing before releases. gemini_update is still limited to master as the most recent version.
6. Update cyvcf so it doesn't error when AD tag is used for non-list data.
7. Fix regression in cyvcf to handle Flags in info field. (Thanks to Jon for reporting)
8. Improvements to install related to PYTHONHOME and other env variables(@chapmanb & @bw2)

0.12.2

Corrected a stale .c file in the cyvcf library. This is effectively a replacement for the 0.12.1 release.

0.12.1

1. Support for input VCF files containing variants with multiple alternate alleles. Thanks to Brent Pedersen.
2. Updated, decomposed, and normalized the ExAC, Clinvar, Cosmic, dbSNP, and ESP annotation files to properly support variants with multiple alternate alleles.
3. Updated the logic for the clinvar significance column to retain all documented singificances.
4. Support for VCF annotation files in the *annotate* tool.
5. Improved the speed of loading by 10-15%. Thanks to Brent Pedersen.
6. Added *–only-affected* and *–min-kindreds* options to the compound heterozygotes tool.
7. Added a *–format vcf* option to the *query* tool to output query results in VCF format.
8. Added the *–families* option to the *auto_**, *de_novo*, and *comp_hets* tools. Thanks to Mark Cowley and Tony Roscioli.
9. Added the *–only-affected* option to the *de_novo* tool.
10. Allow the *–sample-filter* to work with *–format TPED*. Thanks to Rory Kirchner.
11. Add *–format sampledetail* option that provides a melted/tidy/flattened version of samples along with *–showsample* and includes information from samples table. Thanks to Brad Chapman.
12. Add 'not' option to –in filtering. Thanks to Rory Kirchner.
13. Fixed a bug in the *de_novo* tool that prevented proper function when families have affected and unaffected children. Thanks to Andrew Oler.
14. Fixed a bug in cyvcf that falsely treated '.|.' genotypes as homozygous alternate. Thanks to Xiao Xu.
15. GEMINI now checks for and warns of old grabix index files. Thanks to Andrew Oler and Brent Pedersen.
16. Fixed a bug that added newlines at the end of tab delimited PED files. Thanks to Brad Chapman.

0.11.0

1. Integration of ExAC annotations (v0.2): http://exac.broadinstitute.org/
2. New tools for cancer genome analysis. Many thanks to fantastic work from Colby Chiang.
   - *gemini set_somatic*
   - *gemini actionable_mutations*
   - *gemini fusions*
3. Improved support for structural variants. New columns include:
   - *sv_cipos_start_left*
   - *sv_cipos_end_left*
   - *sv_cipos_start_right*
   - *sv_cipos_end_right*
   - *sv_length*
   - *sv_is_precise*
   - *sv_tool*
   - *sv_evidence_type*
   - *sv_event_id*
   - *sv_mate_id*
   - *sv_strand*
4. Updated the 1000 Genomes annotations to the Phase variant set.
5. Added *clinvar_causal_allele* column.
6. Fixed a bug in grabix that caused occasional duplicate and missed variants.

0.10.1

1. Add *fitCons <http://biorxiv.org/content/early/2014/09/11/006825>* scores as an additional measure of potential function in variants of interest, supplementing existing CADD and dbNSFP approaches.
2. Updated Clinvar, COSMIC, and dbSNP to their latest versions.

0.10.0

1. Provide an `--annotation-dir` argument that specifies the path the annotation databases, to overwrite configured data inputs. Thanks to Björn Grüning,
2. Support reproducible versioned installs of GEMINI with Python dependencies. Enables Galaxy integration. Thanks to Björn Grüning,

0.8.0

1. Support arbitrary annotation supplied to VEP, which translate into queryable columns in the main variant table.
2. Improve the power of the genotype filter wildcard functionality.

0.7.1

1. Suppress openpyxl/pandas warnings (thanks to @chapmanb)
2. Fix unit tests to account for cases where a user has not downloaded the CADD or GERP annotation files. Thanks to Xialoin Zhu and Daniel Swensson for reporting this and to Uma Paila for correcting it.

0.7.0

1. Added support for CADD scores via new `cadd_raw` and `cadd_scaled` columns.
2. Added support for genotype wildcards in query select statements. E.g., `SELECT chrom, start, end (gts).(phenotype==2) FROM variants`. See http://gemini.readthedocs.org/en/latest/content/querying.html#selecting-sample-genotypes-based-on-wildcards.
3. Added support for genotype wildcards in the –gt-filter. E.g., `--gt-filter "(gt_types).(phenotype==2).(==HET).` See http://gemini.readthedocs.org/en/latest/content/querying.html#gt-filter-wildcard-filtering-on-genotype-columns.
4. Added support for the VCF INFO field both in the API and as a column that can be SELECT'ed.
5. Upgraded to the latest version of ClinVar.
6. Standardized impacts to use Sequence Ontology (SO) terms.
7. Automatically add indexes to custom, user-supplied annotation columns.
8. Improvements to the installation script.
9. Fixed bugs in the handling of ClinVar UTF8 encoded strings.
10. Upgraded the `gene_summary` and `gene_detailed` tables to version 75 of Ensembl.
11. Added support for the MPI Mouse Phenotype database via the `mam_phenotype_id` column in the `gene_summary` table.
12. Enhanced security.
13. Corrected the ESP allele frequencies to be based report _alternate_ allele frequency instead of _minor_ allele frequency.
14. VEP version support updated (73-75) Support for aa length and bio type in VEP.
15. The *lof_sieve* tool support has been extended to VEP annotations.
16. Added the `ccds_id` and `entrez_id` columns to the `gene_detailed` table.

0.6.6

1. Added COSMIC mutation information via new cosmic_ids column.

0.6.4 (2014-Jan-03)

1. New annotation: experimentally validated human enhancers from VISTA.
2. Installation improvements to enable isolated installations inside of virtual machines and containers without data. Allow data-only upgrades as part of `update` process.
3. Fix for gemini query error when `--header` specified (#241).

0.6.3.2 (2013-Dec-10)

1. Fixed a bug that caused `--gt-filter` to no be enforced from `query` tool unless a GT* column was selected.
2. Support for ref and alt allele depths provided by FreeBayes.

0.6.3.1 (2013-Nov-19)

1. Fixed undetected bug preventing the `comp_hets` tool from functioning.
2. Added unit tests for the `comp_hets` tool.

0.6.3 (2013-Nov-7)

1. Addition permutation testing to the c-alpha test via the `--permutations` option.
2. Addition of the `--passonly` option during loading to filter out all variants with a filter flag set.
3. Fixed bug with parallel loading using the extended sample table format.
4. SLURM support added.
5. Refactor of loading options to remove explosion of xxx-queue options. Now load using `--scheduler` on `--queue`.
6. Refactor of Sample class to handle the expanded samples table.
7. Addition of `--carrier-summary-by-phenotype` for summarizing the counts of carriers and non-carriers stratified by the given sample phenotype column.
8. Added a `--nonsynonymous` option to the C-alpha test.
9. Added `gemini amend` to edit an existing database. For now only handles updating the samples table.
10. Fixed a bug that prevented variants that overlapped with multiple 1000G variants from having AAF info extracted from 1000G annotations. This is now corrected such that multiple overlaps with 1000G variants are tolerated, yet the logic ensures that the AAF info is extracted for the correct variant.
11. Fixed installation issues for the GEMINI browser.
12. Added `--show-families` option to gemini query.

0.6.2 (2013-Oct-7)

1. Moved *–tped* and *–json* options into the more generic *–format* option.
2. Fixed bug in handling missing phenotypes in the sample table.
3. Fixed *–tped* output formatting error.
4. API change: GeminiQuery.run takes an optional list of predicates that a row must pass to be returned.
5. *–sample-filter* option added to allow for restricting variants to samples that pass the given sample query.
6. ethnicity removed as a default PED field.
7. PED file format extended to allow for extra columns to be added to the samples table under the column named in the header.
8. The autosomal_recessive and autosomal_dominant tools now warn, but allow for variants to be detected in the absence of known parent/child relationships.

0.6.1 (2013-Sep-09)

1. Corrected bug in de_novo tool that was undetected in 0.6.0. Unit tests have been added to head this off in the future. Thanks to **Jessica Chong**
2. Added the *-d* option (minimum sequence depth allowed for a genotype) to the *autosmal_recessive* and *autosmal_dominant* tools.
3. New *–tped* option in the *query* tool for reporting variants in TPED format. Thanks to **Rory Kirchner**.
4. New *–tfam* option in the *dump* tool for reporting sample infor in TFAM format. Thanks to **Rory Kirchner**.

0.6.0 (2013-Sep-02)

1. Add the `--min-kindreds` option to the `autosomal_recessive` and `autosomal_dominant` tools to restrict candidate variants/genes to those affecting at least `--min-kindreds`. Thanks to **Jessica Chong**
2. Addition of a new `burden` tool for gene or region based burden tests. First release supports the C-alpha test. Thanks to **Rory Kirchner**.
3. Use of Continuum Analytics Anaconda python package for the automated installer. Thanks to **Brad Chapman**.
4. Enhancements to the `annotate` tool allowing one to create new database columns from values in custom BED+ annotation files. Thanks to **Jessica Chong** and **Graham Ritchie**.
5. Addition of the `--column`, `--filter`, and `--json` options to the `region` tool.
6. Improvements to unit tests.
7. Allow alternate sample delimiters in the `query` tool via the `--sample-delim` option. Thanks to **Jessica Chong**.
8. Provide a REST-like interface to the gemini browser. In support of future visualization tools.
9. Allow the `query` tool to report results in JSON format via the `--json` option.
10. Various minor improvements and bug fixes.

0.5.0b (2013-Jul-23)

1. Tolerate either -9 or 0 for unknown parent or affected status in PED files.
2. Refine the rules for inheritance and parental affected status for autosomal dominant inheritance models.
3. The `autosomal_dominant`, `autosomal_recessive`, and `de_novo` mutation tools have received the following improvements.
    - improved speed (especially when there are multiple families)
    - by default, all columns in the variant table are reported and no conditions are placed on the returned variants. That is, as long as the variant meets the inheritance model, it will be reported.
    - the addition of a `--columns` option allowing one to override the above default behavior and report a subset of columns.
    - the addition of a `--filter` option allowing one to override the above default behavior and filter reported variants based on specific criteria.

4. The default minimum aligned sequencing depth for each variant reported by the `de_novo` tool is 0. Greater stringency can be applied with the `-d` option.

0.4.0b (2013-Jun-12)

1. Added new `gt_ref_depths`, `gt_alt_depths`, and `gt_quals` columns.
2. Added a new `--show-samples` option to the `query` module to display samples with alternate allele genotypes.
3. Improvements and bug fixes for installation.

0.3.0b

1. Improved speed for adding custom annotations.
2. Added GERP conserved elements.
3. Optionally addition of GERP conservation scores at base pair resolution.
4. Move annotation files to Amazon S3.

F.A.Q.

Does GEMINI work with non-human genomes?

Currently, no. However, we recognize that the GEMINI framework is suitable to genetic research in other organisms. This may be a focus of future work.

What versions of the human genome does GEMINI support?

Currently, we support solely build 37 of the human genome (a.k.a, hg19). We intend to support forthcoming versions of the human genome in future releases.

How can I use PLINK files with GEMINI?

Many datasets, especially those derived from GWAS studies, are based on SNP genotyping arrays, and are thus stored in the standard PLINK formats. While GEMINI only supports VCF input files, it is relatively straightforward to convert PLINK datasets to VCF with the PLINK/SEQ toolkit.

1. First, load the PLINK BED file into a new PLINK/SEQ project using the instructions found in the "Load a PLINK binary fileset" section here.

2. Next, use PLINK/SEQ to convert the project to VCF using the instructions found here.

At this point, you should have a VCF file that is compatible with GEMINI.

Alternatively, in his bcbio project, Brad Chapman has written a convenient script for directly converting PLINK files to VCF. Below is an example of how to use this script.

```
$ plink_to_vcf.py <ped file> <map file> <UCSC reference file in 2bit format)
```

Other information

Besides the GEMINI manuscript itself, see this presentation for a high-level overview of GEMINI.

Additionally, this video provides more details about GEMINI's aims and utility.