Pipeline Example: Compound Heterozygous Variant Identification

Dustin Miller

3/10/2020

Introduction

This example takes users through the steps needed to take a VCF or gVCF file and identify Compound Heterozygous (*CH*) variants. This pipeline is generalized and depending on the data state, some pipeline steps may not be necessary. The example data used throughout this pipeline is from Genome in a Bottle, for an Ashkenazi Trio (A trio is data from the affected child and both parents). The files are in VCF format and were aligned to GRCh38. However, this pipeline can also process gVCF or VCF files aligned to GRCh37.

The Dockerfile and all python scripts are available at github.com/dmiller903/ch-pipeline. Clone this repository on the system where you be processing your data. Place this file where it can be accessed by your docker container.

The steps of this pipeline assume that you have already installed Docker on the system where you will be processing your data. If you have not installed Docker, see the Docker Engine overview for installation instructions.

Set-up

Change root.dir to the path below where you cloned the ch-pipeline. For example, if you cloned the repository to "/example/ch-pipeline", set root.dir to "/example".

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_knit$set(root.dir="/example")
```

Build Docker Image

All steps of the pipeline will use the same docker image. The docker image has all the necessary tools for any given step of the pipeline to complete. To build the image, the "docker build" command requires that you be in the folder where the Dockerfile is located. Use the "docker build" command below. "-q" runs the build command in quiet mode and will print image ID on successful completion. "-t" is used to name the image, in this case, the image is being named "parse". The "." tells Docker to use the Dockerfile that is in the directory.

```
cd ch-pipeline/docker-files/CompoundHetVIP
docker build -q -t compound-het-vip .
cd ../../..
```

Download VCF File for Each Member of the Trio

As mentioned in the Introduction, the VCF files that need to be downloaded for this example are for an Ashkenazi Trio

```
# Create a directory for the Genome in a Bottle data in the directory root.dir was set to
mkdir example/giab
# Download vcf for father
wget -nv ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/\
HG003_NA24149_father/latest/GRCh38/HG003_GRCh38_GIAB_highconf_CG-Illfb-\
IllsentieonHC-Ion-10XsentieonHC_CHROM1-22_v.3.3.2_highconf.vcf.gz \
-O giab/giab_father.vcf.gz

# Download vcf for mother
wget -nv ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/\
HG004_NA24143_mother/latest/GRCh38/HG004_GRCh38_GIAB_highconf_CG-Illfb-\
IllsentieonHC-Ion-10XsentieonHC_CHROM1-22_v.3.3.2_highconf.vcf.gz \
-O giab/giab_mother.vcf.gz

# Download vcf for child
wget -nv ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/\
HG002_NA24385_son/latest/GRCh38/HG002_GRCh38_GIAB_highconf_CG-Illfb-IllsentieonHC-\
Ion-10XsentieonHC-SOLIDgatkHC_CHROM1-22_v.3.3.2_highconf_triophased.vcf.gz \
-O giab/giab_child.vcf.gz
```

Keep variant-only sites of VCF or gVCF files

This example uses a VCF, but a gVCF may also be used. If input files are VCF and no parent files are available, then this script does not need to be used. However, if files are VCF files and parent files are available, the script will filter each parent gVCF file for sites that only occur in the child of that family.

gVCF files are different from VCF files in that they contain information for every position, including non-variant positions. Therefore, these files are large and would take a long time to process if the non-variant positions were included throughout the whole compound heterozygous pipeline. Therefore, if gVCF files are used, this step takes a patient gVCF file and removes all non-variant sites. It outputs the file to a path specified by the user. If parent files for the patient are included, the script will filter each parent gVCF file for sites that only occur in the patient of that family.

Execute Script

The "keep_variant_sites.py" script is needed for this step and takes 2 required arguments: 1) the input sample gVCF or VCF file 2) the path where the output should be saved. The optional arguments are: 1) –parent_1_file, which is the maternal or paternal gVCF or VCF of the sample 2) –parent_2_file, which is the other maternal or paternal gVCF or VCF of the sample. 3) –output_suffix which is the suffix for each output file (do not include .gz at end as this will be included when the file is bgzipped). The default is set to "_parsed.vcf" 4) –is_gvcf which indicates whether or not a gVCF file is used. If argument is set to "y", all non-variant sites will be filtered out of the sample file and a new VCF will be created.

The "docker run" command used for this and subsequent steps use various options. The "-d" option allows the container to run in the background and will save a log where you direct it to with ">" as seen below. The "-v" option allows you to mount a volume from your local or remote machine to the container. For example, "-v /example:/proj" allows the container to access the "example" directory and all sub directories and is known as "/proj" in the container. "-w" sets the working directory inside the container. At the end of the docker run command, "parse" is the name of the image to execute as a container. Everything after this line, up until ">", will be executed in the container.

```
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/keep_variant_sites.py \
```

```
  giab/giab_child.vcf.gz \
  giab/ \
  --parent_1_file giab/giab_father.vcf.gz \
  --parent_2_file giab/giab_mother.vcf.gz \
  --is_gvcf n \
  > giab/keep_variant_sites.out

#This is not necessary but allows you to see any output log information in real-time
CONTAINERID="cat giab/keep_variant_sites.out"
docker logs -f `$CONTAINERID`
```

Combine each trio into a single file

This script is only necessary if you are have trio data (data from child and both parents). During this step, the VCF of the parents and child will be combined into a single VCF file. This is done so that later, when phasing, a single trio file can be phased.

Set a new empty directory where reference files are to be downloaded

The script used below executes GATK4 to combine gVCF files of trios, or bcftools to combine VCF files of trios. GATK4 requires reference files and bcftools does not. If using gVCF files, the script will install necessary reference files in the path you indicate with "-v". This example does not use gVCF files, but if it did, you would create a new empty sub directory in the directory you set root.dir. For example:

```
mkdir /example/references
```

Execute Script

The "combine_trio.py" script is needed for this step and takes 4 arguments: 1) sample VCF or gVCF file 2) maternal or paternal VCF or gVCF file 3) other maternal or paternal VCF or gVCF file 4) the name of the combined output file. An optional argument –is_gvcf, indicates if the files were originally gVCF or not. The default is set to yes, "y". For this example, this argument is set to "n".

```
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/combine_trio.py \
  giab/giab_child.vcf.gz \
  giab/giab_father_parsed.vcf.gz \
  giab/giab_mother_parsed.vcf.gz \
  giab/giab_trio.vcf \
  --is_gvcf n \
  > giab/combine_trio.out

CONTAINERID="cat giab/combine_trio.out"
docker logs -f `$CONTAINERID`
```

If gVCF files are being use, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "references" in my "/example" path. In this case, another "-v" needs to be added, "-v /example/references:/references". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/references" for the script to work properly. Notice in the below example, another -v has been added, and –is_gvcf has been changed to "y".

```
docker run -d -v /example:/proj -v /example/references:/references \
  -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/combine_trio.py \
  giab/giab_child.gvcf.gz \
  giab/giab_father_parsed.gvcf.gz \
  giab/giab_mother_parsed.gvcf.gz \
  giab/giab_trio.vcf \
  --is_gvcf y \
  > giab/combine_trio.out

CONTAINERID="cat giab/combine_trio.out"
docker logs -f `$CONTAINERID`
```

Liftover trio files and individual files from GRCh38 to GRCh37

Some phasing programs and other programs such as GEMINI, require that VCF or gVCF files be aligned to GRCh37. Therefore, this step takes an input VCF that is aligned to GRCh38 converts it to GRCh37 positions.

Use the previously defined reference file directory or set a new empty directory where reference files are to be downloaded

The script used below executes Picard's Liftover tool to convert the file from GRCh38 to GRCh37. Picard requires reference files to accomplish liftover. If not already done above, create a new empty sub directory in the directory you set root.dir. For example:

```
mkdir /example/references
```

If you already did this above, you can use the same directory you created.

Execute Script

The "liftover.py" script is needed for this step and takes 2 arguments: 1) an input VCF file 2) the name of the output VCF file. Another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "references" in my "/example" path. In this case, another "-v" needs to be added, "-v /example/references:/references". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/references" for the script to work properly.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/liftover.py \
  giab/giab_trio.vcf \
  giab/giab_trio_GRCh37.vcf \
  > giab/liftover.out

CONTAINERID="cat giab/liftover.out"
docker logs -f `$CONTAINERID`
```

Remove unplaced, and multiallelic sites and duplicate sites from lifted files

During liftover, some randomly placed sites are included in the VCF file. These randomly placed sites are those that are in GRCh38 but the exact position in GRCh37 isn't known. Therefore, for subsequent analysis, these sites are removed. Only sites with known positions, on a known chromosome are kept. In addition, positions that are multiallelic or are duplicates are removed because programs such as PLINK (used next) and SHAPEIT2 (used later) can not handle these types of sites. Sites that contain any missing genotype information (i.e. "./.") are removed to improve phasing accuracy.

This step does not require an additional image if all previous steps above have been followed. If previous steps were not followed, see the parse docker image as explained above.

Execute Script

The "remove_unplaced_multiallelic.py" script is needed for this step and takes 2 arguments: 1) the input VCF file and the 2) the output VCF file (do not include .gz at end of file name, the script will bgzip file at end and add ".gz") If you want to keep positions where "./." is found, use the optional argument, –remove_unknown_genotypes, and set it to "n".

```
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/remove_unplaced_multiallelic.py \
  giab/giab_trio_GRCh37.vcf.gz \
  giab/giab_trio_GRCh37_no_unplaced_or_duplicates.vcf \
  > giab/remove_unplaced_multiallelic.out

CONTAINERID="cat giab/remove_unplaced_multiallelic.out"
docker logs -f `$CONTAINERID`
```

Separate VCF file into chromosome files, then generate plink files for each chromosome file

Phasing programs require that chromosomes be phased separately. Some phasing programs, such as SHAPEIT2, require PLINK files in order to phase. Therefore, this script separates a VCF into chromosome VCF files. This step also generates the necessary PLINK files needed for phasing (bed, bim, fam).

Execute Script

The "separate_chr_generate_plink.py" script is needed for this step and takes 2 required arguments: 1) input VCF file 2) the prefix name of output files (no suffix, e.g. "/Data/file1"). If using a trio, the optional argument, –fam_file, needs to be used to create appropriate PLINK files. If no fam file is included, PLINK will output a generic fam file.

```
# Example of how to create fam file
touch giab/giab_trio.fam
echo "giab_trio    HG002    HG003    HG004    1    2
giab_trio    HG003    0    0    1    1
giab_trio    HG004    0    0    2    1" > giab/giab_trio.fam

# Execute container
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/separate_chr_generate_plink.py \
  giab/giab_trio_GRCh37_no_unplaced_or_duplicates.vcf.gz  \
  giab/giab_GRCh37  \
```

```
  --fam_file giab/giab_trio.fam \
  > giab/separate_chr_generate_plink.out

CONTAINERID="cat giab/separate_chr_generate_plink.out"
docker logs -f `$CONTAINERID`
```

Phase each of the trios with a haplotype reference panel using SHAPEIT2, Beagle, or Eagle

This pipeline can phase with SHAPEIT2, Eagle2, and Beagle. All programs will integrate 1000 Genomes Project phase 3 haplotype reference panel. The files used for phasing can be found at https://mathgen.stats.ox.ac.uk/impute/1000GP_Phase3.html and ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/. Files from these sites will be downloaded to your local/remote machine, to the path you designate, the first time one of the phasing scripts is ran.

Use the previously defined reference file directory or set a new empty directory where reference files are to be downloaded

If not already done above, create a new empty sub directory in the directory you set root.dir. For example:

```
mkdir /example/references
```

If you already did this above, you can use the same directory you created.

Phase with SHAPEIT2

The default parameters for phasing are set so that SHAPEIT2 uses family relationship genotype information and also uses a haplotype reference panel. SHAPEIT2 needs to phase each chromosome separately. So this part can be done one chromosome at a time, or scripted so that the user does not need to input each individual chromosome.

The "phase_with_shapeit.py" script is needed for this step and takes 3 arguments: 1) the path and name of the files (Either plink files (bed, bim, fam) or a VCF file. Plink files should be used if a trio and no suffix should be included in file name (e.g. giab/giab_GRCh35_chr21). For non-trios, VCF can be used as input, and any file suffixes should be included (e.g. giab/giab_GRCh37_chr21.vcf.gz)) 2) The output path and name without suffix. 3) the chromosome number to be phased. If the input file used is not a trio, then the optional parameter "–is_trio" can be set to "n". If "–is_trio" is set to "n", then the input file needs to be a VCF file (include suffix). In the docker command, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "references" in my "/example" path. In this case, another "-v" needs to be added, "-v /example/references:/references". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/references" for the script to work properly.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/phase_with_shapeit.py \
  giab/giab_GRCh37_chr21 \
  giab/giab_GRCh37_chr21_shapeit \
  21 \
  > giab/giab_GRCh37_chr21_shapeit.out
```

```
CONTAINERID="cat giab/giab_GRCh37_chr21_shapeit.out"
docker logs -f `$CONTAINERID`
```

If you want to script this out, you could do something like:

```
import os
for i in range(1, 23):
  os.system(f"docker run -d -v /example:/proj -v /example/references:/references \
  -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/phase_with_shapeit.py \
  giab/giab_GRCh37_chr{i} \
  giab/giab_GRCh37_chr{i}_shapeit \
  {i} \
  > giab/phase_with_shapeit_chr{i}.out")
```

Phase with Beagle

The parameters for phasing are set so that Beagle uses a haplotype reference panel. No family relationships are considered. The "phase_with_beagle.py" script is needed for this step and takes 3 arguments: 1) the path and name of the vcf file 2) The output path and name without suffix 3) the chromosome number to be phased. This script has not optional parameters. In the docker command, another "-v" needs to be set to where you want the reference files saved. See "Phase with SHAPEIT2" subheading for an explanation of "-v" usage.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/phase_with_beagle.py \
  giab/giab_GRCh37_chr21.vcf.gz \
  giab/giab_GRCh37_chr21_beagle_phased \
  21 \
  > giab/phase_with_beagle_chr21.out

CONTAINERID="cat giab/phase_with_beagle_chr21.out"
docker logs -f `$CONTAINERID`
```

See "Phase with SHAPEIT2" for an example of how this could be scripted out to do multiple chromosomes with a single command.

Phase with Eagle2

The parameters for phasing are set so that Eagle uses a haplotype reference panel. No family relationships are considered. The "phase_with_eagle.py" script is needed for this step and takes 3 arguments: 1) the path and name of the vcf file 2) The output path and name without suffix 3) the chromosome number to be phased. This script has not optional parameters. In the docker command, another "-v" needs to be set to where you want the reference files saved. See "Phase with SHAPEIT2" subheading for an explanation of "-v" usage.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/phase_with_eagle.py \
```

```
  giab/giab_GRCh37_chr21.vcf.gz \
  giab/giab_GRCh37_chr21_eagle_phased \
  21 \
  > giab/phase_with_eagle_chr21.out

CONTAINERID="cat giab/phase_with_eagle_chr21.out"
docker logs -f `$CONTAINERID`
```

See "Phase with SHAPEIT2" for an example of how this could be scripted out to do multiple chromosomes with a single command.

Revert REF/ALT to be congruent with reference panel

The phased results from SHAPEIT2 can have the REF and ALT alleles switched as compared to the reference genome. We are unsure exactly why this occurs. For files with trios, it may be that since each file only has 3 samples, the ALT allele is more common in the trio and becomes the REF. This step ensures that the REF/ALT alleles of the phased VCF files are congruent with the REF/ALT of the reference genome. In addition, sites with Mendel errors are removed.

This step uses the Docker Image from the previous step, "phase-pipeline".

Execute Script

The "alt_ref_revert.py" script is needed for this step and takes 3 arguments: 1) the input phased VCF file 2) the output VCF file name 3) the chromosome number which is needed so the script can determine which reference file to use. This step assumes you have phased with SHAPEIT2 and that reference files have already been downloaded. In the docker command, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "references" in my "/example" path. In this case, another "-v" needs to be added, "-v /example/references:/references". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/references" for the script to work properly.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/alt_ref_revert.py \
  giab/giab_GRCh37_chr21_shapeit.vcf.gz \
  giab/giab_GRCh37_chr21_shapeit_reverted.vcf \
  21 \
  > giab/alt_ref_revert_shapeit_chr21.out

CONTAINERID="cat giab/alt_ref_revert_shapeit_chr21.out"
docker logs -f `$CONTAINERID`
```

See "Phase with SHAPEIT2" for an example of how this could be scripted out to do multiple chromosomes with a single command.

Concat and merge phased trio chromosome files into one VCF file

To make subsequent analysis of the phased files easier, this step concatenates all phased chromosomes into a single file. If you have multiple sample files (e.g. trio1, trio2, trio3, etc.), this script can merge these files into one after concatenating chromosomes together.

This step uses a previously created docker image "combine-trio". See the steps above to create this container if not done so already.

Execute Script

The "concat_merge_phased_vcf.py" script is needed for this step and takes 2 arguments: 1) A path where all the phased files are stored. Only phased files from a single phasing method should be in this folder. Also, if you have .fam files and you are merging different families into one file, provide .fam files in this folder as well. 2) The name of the output file. If you are merging multiple VCF files into one after concatenating chromosomes, then use the optional argument "–merge_files" and set it to "y". Default is "n". If you want a combined .fam file output, then provide a name for this file with the "–output_fam_file" argument.

```
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/concat_merge_phased_vcf.py \
  giab/phased_files/ \
  giab/giab_combined.vcf \
  --output_fam_file giab/combined_fam.fam \
  > giab/concat_merge_phased_vcf.out

CONTAINERID="cat giab/concat_merge_phased_vcf.out"
docker logs -f `$CONTAINERID`
```

Trim and normalize VCF file

Prior to annotation and loading a database into GEMINI (used later for genotype queries), GEMINI recommends left trimming and normalizing VCF files. Files that are not left trimmed and normalized can be annotated incorrectly and may be incorrectly handled by GEMINI. This step uses VT tools (using a python script) to trim and normalize the phased VCF file. VT tools uses references files and these sites will be downloaded to your local/remote machine, to the path you designate, the first time one of the phasing scripts is ran.

Use the previously defined reference file directory or set a new empty directory where reference files are to be downloaded

If not already done above, create a new empty sub directory in the directory you set root.dir. For example:

```
mkdir /example/references
```

If you already did this above, you can use the same directory you created.

Execute Script

The "vt_split_trim_left_align.py" script is needed for this step and requires 2 arguments: 1) input VCF name 2) output VCF name. In the docker command, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "references" in my "/example" path. In this case, another "-v" needs to be added, "-v /example/references:/references". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/references" for the script to work properly.

```
docker run -d -v /example:/proj -v /example/references:/references -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/vt_split_trim_left_align.py \
  giab/giab_combined.vcf.gz \
  giab/giab_combined_vt.vcf \
  > giab/vt_split_trim_left_align.out

CONTAINERID="cat giab/vt_split_trim_left_align.out"
docker logs -f `$CONTAINERID`
```

Annotate with snpEff

snpEff is used to annotate. Annotation provides information on the effects of variants on known genes. GEMINI allows users to explore genetic variation based on the annotations of the genome.

Create a new, empty, sub directory within the references path created earlier

Using the same path set up earlier for references, add a sub directory to this path, such as "snpEff_data". This will help keep files organized but isn't necessary.

```
mkdir /example/references/snpEff_data
```

Execute Script

The "annotate.py" script is needed for this step and requires 2 arguments: 1) name of input VCF 2) name of annotated VCF file. snpEff requires annotation reference files in order to annotate. These will be downloaded when the "annotate.py" script is executed for the first time. In the docker command, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "snpEff_data" in my "/example/references" path. In this case, another "-v" needs to be added, "-v /example/references/snpEff_data:/snpEff/./data/GRCh37.75". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/snpEff/./data/GRCh37.75" for the script to work properly.

```
docker run -d -v /example:/proj \
  -v /example/references/snpEff_data:/snpEff/./data/GRCh37.75 -w /proj \
  -t compound-het-vip \
  python3 ch-pipeline/scripts/annotate.py \
  giab/giab_combined_vt.vcf \
  giab/giab_annotated.vcf \
  > giab/annotate.out

CONTAINERID="cat giab/annotate.out"
docker logs -f `$CONTAINERID`
```

Load VCF as GEMINI database and query for *CH* variants

GEMINI is used to query the VCF file for *CH* variants. GEMINI requires that the input VCF file be loaded as a GEMINI database. Once in a GEMINI database, various queries can be used to analyze variant data. Our scripts are tailored to identifying *CH* and *de novo* variants.

Create a new, empty, sub directory within the references path created earlier

Using the same path set up earlier for references, add a sub directory to this path, such as "gemini_data". This will help keep files organized but isn't necessary.

```
mkdir /example/references/gemini_data
```

Execute Script

The "gemini_load.py" script is needed for this step and requires 2 arguments: 1) name of annotated VCF file 2) Name of output database (name needs to end in .db). The optional argument "–fam_file" should be used if your VCF file contains family samples. GEMINI will use this file to keep track of family relationships and this allows for *CH* and *de Novo* variant identification. GEMINI databases can take a while to be created, but can be significantly sped up with the "–num_cores" argument. The default is 2. In order to load a GEMINI database, annotation files are necessary. In addition, CADD scores are required by us, not GEMINI, because many of our queries involve CADD scores. These files will be downloaded the first time the "gemini_load.py" script is executed first time. In the docker command, another "-v" needs to be set to where you want the reference files saved. Above I created the sub directory "gemini_data" in my "/example/references" path. In this case, another "-v" needs to be added, "-v /example/references/gemini_data:/usr/local/share/gemini/gemini_data". The portion before the colon (:) is where the files will be saved and the portion after the colon is where that path is located in the container and this portion NEEDS to be "/usr/local/share/gemini/gemini_data" for the script to work properly.

```
docker run -d -v /example:/proj \
  -v /example/references/gemini_data:/usr/local/share/gemini/gemini_data \
  -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/gemini_load.py \
  giab/giab_annotated.vcf \
  giab/giab.db \
  --fam_file giab/combined_fam.fam \
  --num_cores 42 \
  > giab/gemini_load.out

CONTAINERID="cat giab/gemini_load.out"
docker logs -f `$CONTAINERID`
```

To perform a GEMINI query, use "gemini_query.py". This script has 2 arguments: 1) name of GEMINI database to query 2) the output name prefix for query results. There are numerous ways one could query a gemini database. In the "gemini_query.py" script, we query for *CH* and *de novo* variants using various filters. Feel free to only use the queries in the script, but if those queries don't fit your needs, feel free to modify the script or create your own for additional queries. See GEMINI's documentation for additional query options.

```
docker run -d -v /example:/proj -w /proj -t compound-het-vip \
  python3 ch-pipeline/scripts/gemini_query.py \
  giab/giab.db \
  giab/giab \
  > giab/gemini_query.out

CONTAINERID="cat giab/gemini_load.out"
docker logs -f `$CONTAINERID`
```

Note: only the output file ending in "_ch_no_filter.tsv" will contain output. For this particular dataset, no compound heterozygous variants are found with the other queries.