# Querying Variants with the Command Line

Table of Contents:

## Overview

There are two possible ways of querying variants in OpenCGA using the Command Line Interface (CLI), these are:

- **opencga.sh:** this is the user command line, it works **remotely** (outside of OpenCGA cluster) by querying the REST or gRPC services. This can also query Catalog data.
- **opencga-analysis.sh**: a *private* and *internal* command line, this is **not intended** to be used by users and it **only** works inside the OpenCGA cluster.

Although both command lines provide similar functionality users are expected to use opencga.sh. They can be found in the _bin_ folder of OpenCGA installation directory.

## Using *opencga.sh*

This allows to query by: *genomic regions and feature IDs such as gene and SNPa* query by variant annotation such as consequence types, conservations scores, polyphen, sift or population frequencies *sa mple genotypes* variant stats in the study * some basic aggregations such as ranks, group-by or counts

All these filters can be combined. There are some query modifiers implemented: *skip and limit* count: this can be added to **all** CLIs and return the number of results

From the *$OPENCGA*HOME_ folder you can execute to see all the parameters:

```
./bin/opencga.sh variants query -h
```

**NOTE**: for security reasons you need to login into OpenCGA if you want to use this CLI in a standard OpenCGA installation, this will guarantee you only access to the data you have permission, to login you only need to execute:

```
./bin/opencga.sh users login -u USER -p PASSWORD
```

A session token will be stored in your home directory and used internally by OpenCGA Storage.

The command line implements many filters which allows a powerful and highly flexibility queries, including genomic regions, feature IDs (e.g. gene and SNP ids), consequence types, conservation scores, polyphen, sift, population frequencies, ... and even some basic aggregations such as ranks, group-by or counts. All these filters can be combined. There are also some query modifiers implemented: *include, exclude, skip, limit* and *count*, which can be added to most queries.

You can execute *opencga.sh* to see all the parameters. **Please note that *opencga.sh* script is located within the *opencga/bin*** directory in the installation directory. You can see an integrated help with *-h* (or -*-help*) parameter, you can see this by expanding next section:

**opencga.sh help usage**

```
$ cd opencga
$ ./bin/opencga.sh variant query -h

Usage:   opencga.sh variant query [options]

Options:
      --apf, --alt-population-frequency STRING     Alternate Population
Frequency: {study}:{population}[<|>|<=|>=]{number}
          --annot-xref                  STRING     XRef
          --cadd                        STRING     Functional score:
{functional_score}[<|>|<=|>=]{number} e.g. cadd_scaled>5.2,cadd_raw<=0.3
          --clinvar                     STRING     Alias to id
      --ch, --compound-heterozygous     STRING     [PENDING] Take a family
in the form of: FATHER,MOTHER,CHILD and specifies if is affected or not to
                                                   filter by compound
heterozygous, example: 1000g:NA001:aff,1000g:NA002:unaff,1000g:NA003:aff
      -C, --conf                        STRING     Configuration folder
that contains opencga.yml, catalog-configuration.yaml,
                                                   storage-configuration.
yml and client-configuration.yaml files.
      --ct, --consequence-type          STRING     Consequence type SO
term list. example: SO:0000045,SO:0000046
```

```
      -c, --conservation                STRING     Conservation score:
{conservation_score}[<|>|<=|>=]{number} example: phastCons>0.5,phylop<0.1
          --count                                  Total number of
results. Default = false [false]
          --dominant                    STRING     [PENDING] Take a family
in the form of: FATHER,MOTHER,CHILD and specifies if is affected or not to
                                                   filter by dominant
segregation, example: 1000g:NA001:aff,1000g:NA002:unaff,1000g:NA003:aff
          --drug                        STRING     List of drug names
      -E, --exclude                     STRING     Comma separated list of
fields to be excluded from the response
      -f, --file                        STRING     A comma separated list
of files to be used as filter
      -g, --gene                        STRING     List of genes
          --gene-biotype                STRING     Biotype CSV
      --go, --gene-ontology             STRING     List of Gene Ontology
(GO) accessions or names. e.g. "GO:0002020"
          --gene-trait                  STRING     List of gene trait
association IDs or names. e.g. "umls:C0007222,Cardiovascular Diseases"
          --gene-trait-id               STRING     [DEPRECATED] List of
gene trait association names. e.g. "Cardiovascular Diseases"
          --gene-trait-name             STRING     [DEPRECATED] List of
gene trait association id. e.g. "umls:C0007222,OMIM:269600"
      --gt, --genotype                  STRING     A comma separated list
of samples from the SAME study, example: NA0001:0/0,0/1;NA0002:0/1
          --group-by                    STRING     Group by gene, ensembl
gene or consequence_type
      -h, --help                                   Print this help [false]
          --histogram-interval          INT        Histogram interval
size. Default:2000 [0]
          --hpo                         STRING     List of HPO terms. e.g.
"HP:0000545,HP:0002812"
          --id                          STRING     List of variant ids
      -I, --include                     STRING     Comma separated list of
fields to be included in the response
          --limit                       INT        Maximum number of
results to be returned [0]
          --log-file                    STRING     Set the file to write
the log
      -L, --log-level                   STRING     One of the following:
'error', 'warn', 'info', 'debug', 'trace' [info]
      -M, --metadata                               Include metadata
information [false]
          --mode                        STRING     Communication mode.
grpc|rest|auto. [auto]
          --no-header                              Not include headers in
the output (not applicable to json output-format) [false]
      -o, --output                      STRING     Output file. [STDOUT]
      --of, --output-format             STRING     Output format. one of
{JSON, JSON_PRETTY, TEXT, YAML} [TEXT]
          --output-histogram                       Calculate histogram.
Requires --region. [false]
          --output-sample               STRING     A comma separated list
of samples from the SAME study to be returned
          --output-study                STRING     A comma separated list
of studies to be returned
          --output-unknown-genotype     STRING     Returned genotype for
unknown genotypes. Common values: [0/0, 0|0, ./.] [./.]
      --annotations, --output-vcf-info  STRING     Set variant annotation
to return in the INFO column. Accepted values include 'all', 'default' aor
a
                                                   comma-separated list
such as 'gene,biotype,consequenceType'
      --pmaf, --population-maf          STRING     Population minor allele
frequency: {study}:{population}[<|>|<=|>=]{number}
          --protein-keywords            STRING     List of Uniprot protein
keywords
      --ps, --protein-substitution      STRING     Filter by Sift or/and
Polyphen scores, e.g. "sift<0.2;polyphen<0.4"
          --rank                        STRING     Rank variants by gene,
ensemblGene or consequence_type
```

```
          --recessive                   STRING     [PENDING] Take a family
in the form of: FATHER,MOTHER,CHILD and specifies if is affected or not to
                                                   filter by recessive
segregation, example: 1000g:NA001:aff,1000g:NA002:unaff,1000g:NA003:aff
      -r, --region                      STRING     List of regions: {chr}:
{start}-{end}, e.g.: 2,3:1000000-2000000
          --region-file                 STRING     GFF File with regions
          --samples-metadata                       Returns the samples
metadata group by studyId, instead of the variants [false]
      -S, --sid, --session-id           STRING     Token session id, NOTE:
parameter --sid will be delete soon
          --skip                        INT        Number of results to
skip [0]
      --maf, --stats-maf                STRING     Take a <STUDY>:<COHORT>
and filter by Minor Allele Frequency, example: 1000g:all>0.4
      --mgf, --stats-mgf                STRING     Take a <STUDY>:<COHORT>
and filter by Minor Genotype Frequency, example: 1000g:all<=0.4
          --stats-missing-allele        STRING     Take a <STUDY>:<COHORT>
and filter by number of missing alleles, example: 1000g:all=5
          --stats-missing-genotype      STRING     Take a <STUDY>:<COHORT>
and filter by number of missing genotypes, example: 1000g:all!=0
      -s, --study                       STRING     Study [[user@]project:]
study where study and project can be either the id or the alias.
                  --summary                                Fast fetch of
main variant parameters [false]
          --transcript-flag             STRING     List of transcript
annotation flags. e.g. CCDS,basic,cds_end_NF,
                                                   mRNA_end_NF,
cds_start_NF,mRNA_start_NF,seleno
      -t, --type                        STRING     Whether the variant is
a: SNV, INDEL or SV
      -v, --verbose                                Increase the verbosity
of logs [false]
```

## Design considerations

There are some design decisions you must be aware of:

- Comma character ',' is used in different places in the CLI and will always behave as a logical *OR.* For example, in *region 1:1800000-1900000,1:2000000-2100000 or "sift<0.2,polyphen<0.5".* The semi-colon ';' when allowed, will behave as a logical AND.
- Independently where regions, genes or SNPs IDs are in the CLI they always behave as a logical *OR*. For instance in next CLI *region* and *gene* parameters act as a logical *OR*:

```
./opencga.sh variant query --region 1:1849612-1850388,1:2049808-2050192 --
gene BRCA2 --study GONL --exclude studies --of json_pretty
```

- For all the other CLI parameters a logical *AND* is executed, so in next query only variants for the specified regions with a sift below 0.2 AND a polyphen score below 0.5 are returned:

```
./opencga.sh variant query --region 22:17464756-17479892 --protein-
substitution "sift<=0.5,polyphen>=0.1" --study reference_grch38:1kG_phase3
--limit 10 --exclude studies
```

## Example queries

### Using variant attributes

To fetch variants for a specific region:

```
./opencga.sh variant query --studies STUDY --region CHR:START-END
```

For example, to fetch variants from the 1k genomes project on region 22:15000000-20000000:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 22:
15000000-20000000 --limit 3 --exclude studies
```

**Please note:** the number of variants in the region may be huge - hundreds of thousands in the example. The total number of variants returned has been limited to 3 by using the *--limit* parameter. Also, in order to improve the efficiency of the query, all studies metadata, which in turn contain all samples metadata for all 1kG phase 3 samples, are excluded from the result by using the parameter *--exclude*.

To fetch variants from several regions separate them by ',':

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 1:
1800000-1900000,1:2000000-2100000 --limit 3 --exclude studies
```

you can also add a list of genes:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 1:
1800000-1900000,1:2000000-2100000 --gene BRCA2,TP53 --limit 3 --exclude
studies
```

**Note:** remember all regions and genes are always a logical OR.

If you want SNV, INDELS or SV you can use *--type* parameter:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 1:
1800000-1900000,1:2000000-2100000 --limit 3 --exclude studies --type INDEL
```

## Using variant annotation info

To query by SIFT or PolyPhen2 you can use *--sift* and/or *--polyphen*:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 22:
17464756-17479892 --protein-substitution "sift<0.5" --limit 3 --exclude
studies
```

or using both:

```
./opencga.sh variant query --region 22:17464756-17479892 --protein-
substitution "sift<=0.5,polyphen>=0.1" --study reference_grch38:1kG_phase3
--limit 10 --exclude studies
```

To only count the number of variants remember you can always add *--count*:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 22:
17464756-17479892 --protein-substitution "sift>0.5" --count
```

To query using Consequence Type terms from Sequence Ontology (SO), you can use the terms at http://www.ensembl.org/info/genome/variation/predicted_data.html, use comma to add terms:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --consequence-type missense_variant,stop_gained --count
```

To query using conservation scores you can use *--conservation.* Multiple comparisons may be combined by using either the ',' or the ';' as separators. Comparisons separated by ',' will perform an OR logical operation. Comparisons separated by ';' will perform and AND logical operation. Complex logical operations combining ',' and ';' in a single query are not currently allowed. Next query uses both PhastCons and Phylop in separated by ',', since they are different query fields the act as a logical OR:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --conservation "phastCons>0.5,phylop<0.1,gerp>0.1" --
count
```

You can also query using population frequencies from 1000 Genome project, EVS and EXaC using --*population-freqs* parameter:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --alt-population-frequency "1kG_phase3:EUR<0.01" --count
```

or several populations together separated by ',' or ';', since they are different populations and query fields this is a logical OR:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --alt-population-frequency "1kG_phase3:EUR<0.01,1
kG_phase3:AFR<0.01" --count
```

## Sample genotype

To query by specific sample genotypes you can use *--genotype* parameter. You must separate samples by ';', and the accepted genotypes for each sample by ','. This will execute an AND between samples and a OR for the genotypes, so in:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --genotype
"NA19030:0|1,1|0,1|1;NA19043:0|1,1|0,1|1" --limit 3 --exclude studies
```

variants which are present in samples NA19030 and  NA19043 are returned (number of returned variants is limited to 3 in this case)

### Building more complex queries

You can combine all the parameters above to execute more complex queries:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --genotype
"NA19030:0|1,1|0,1|1;NA19043:0|0" --limit 3 --exclude studies,annotation.
geneTraitAssociation --conservation "phastCons<1"
```

### Some aggregations and rankings

To group variants per gene or consequence type you can use *--group-by* parameter:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --group-by gene --include annotation.consequenceTypes --
log-level debug --limit 10
```

You can also rank genes or consequence type using *--rank*:

```
./opencga.sh variant query --study reference_grch37:1kG_phase3 --region 21:
15888971-15889629 --rank gene --include annotation.consequenceTypes --
limit 10
```