Variant Effect Predictor • VEP script

★ Important notice to VEP users

- VEP is now available as the ensembl-vep package from GitHub
- The old version available as part of ensembl-tools will no longer be updated

Use VEP to analyse your variation data locally. No limits, powerful, fast and extendable, the VEP script is the best way to get the most out of VEP and Ensembl.

VEP is a powerful and highly configurable tool - have a browse through the documentation.

★ Quick start

1. Download

```
git clone https://github.com/Ensembl/ensembl-vep.git
```

2. Install

```
cd ensembl-vep
perl INSTALL.pl
```

3. Test

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache
```

You might also like to read up on the data formats that VEP uses, and the different ways you can access genome data. The VEP script can annotate your variants with custom data, be extended with plugins, and use powerful filtering to find biologically interesting results.

Beginners should have a run through the tutorial, or try the web interface first.

If you use VEP in your work, please cite our latest publication **McLaren et. al. 2016** (doi:10.1186/s13059-016-0974-4)

Any questions? Send an email to the Ensembl developers' mailing list or contact the Ensembl Helpdesk.

★ What's new in release 88?

Documentation contents

Download documentation in PDF format

★ Tutorial

Download and install
- Download
- What's new in release 88
- Installation
- Using VEP in Windows

Data formats
- Input
- Output

Running VEP
- Options

Annotation sources
- Caches
- GFF/GTF files
- Databases

Filtering results
- Running filter_vep
- Writing filters

Custom annotations
- Data formats
- Options

Plugins
- Examples
- Using plugins

Examples & use cases
- Example commands
- Citations and VEP users

Other information
- Performance
- Multiple assemblies
- Summarising annotation

- HGVS notations
- RefSeq transcripts

FAQ

- General questions
- Web VEP questions
- VEP script questions

NOTE: If you're using a UNIX or Mac system, you can dive straight into this tutorial by opening your favourite terminal application. If you're on Windows you might like to have a look at the guide for Windows users before starting.

Have you downloaded VEP yet? Use git to clone it:

```
git clone https://github.com/Ensembl/ensembl-vep
cd ensembl-vep
```

VEP uses "cache files" or a remote database to read genomic data. Using cache files gives the best performance - let's set one up using the installer:

```
perl INSTALL.pl
Hello! This installer is configured to install v88 of the Ensembl API for use by VEP.
It will not affect any existing installations of the Ensembl API that you may have.

It will also download and install cache files from Ensembl's FTP server.

Checking for installed versions of the Ensembl API...done
It looks like you already have v88 of the API installed.
You shouldn't need to install the API

Skip to the next step (n) to install cache files

Do you want to continue installing the API (y/n)?
```

If you haven't yet installed the API, type "y" followed by enter, otherwise type "n" (perhaps if you ran the installer before). At the next prompt, type "y" to install cache files

```
Do you want to continue installing the API (y/n)? n
 - skipping API installation

VEP can either connect to remote or local databases, or use local cache files.
Cache files will be stored in /nfs/users/nfs_w/wm2/.vep
Do you want to install any cache files (y/n)? y

Downloading list of available cache files
The following species/files are available; which do you want (can specify multiple separated by
1 : ailuropoda_melanoleuca_vep_88_ailMel1.tar.gz
2 : anas_platyrhynchos_vep_88_BGI_duck_1.0.tar.gz
3 : anolis_carolinensis_vep_88_AnoCar2.0.tar.gz
...
42 : homo_sapiens_vep_88_GRCh38.tar.gz
...

?
```

Type "42" (or the relevant number for homo_sapiens and GRCh38) to install the cache for the latest human assembly. This will take a little while to download and unpack! By default VEP assumes you are working in human; it's easy to switch to any other species using --species [species].

```
? 42
 - downloading ftp://ftp.ensembl.org/pub/release-88/variation/VEP/homo_sapiens_vep_88_GRCh38.ta
 - unpacking homo_sapiens_vep_88_GRCh38.tar.gz

Success
```

By default VEP installs cache files in a folder in your home area (**$HOME/.vep**); you can easily change this using the **-d** flag when running the install script. Have a look at the installer documentation for more details.

VEP needs some input containing variant positions to run. In their most basic form, this should just be a chromosomal location and a pair of alleles (reference and alternate). VEP can also use common formats such as VCF and HGVS as input. Have a look at the

Data formats page for more information.

We can now use our cache file to run VEP on the supplied example file **examples/homo_sapiens_GRCh38.vcf**, which is a VCF file containing variants from the 1000 Genomes Project, remapped to GRCh38:

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache
2013-07-31 09:17:54 - Read existing cache info
2013-07-31 09:17:54 - Starting...
ERROR: Output file variant_effect_output.txt already exists. Specify a different output file
with --output_file or overwrite existing file with --force_overwrite
```

You may see this error message if you've already run the script once. VEP tries not to trample over your existing files unless you tell it to. So let's tell it to using --force_overwrite

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite
```

By default VEP writes to a file named "variant_effect_output.txt" - you can change this file name using -o. Let's have a look at the output that the script generated.

```
head variant_effect_output.txt
## ENSEMBL VARIANT EFFECT PREDICTOR v88.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_88_38 on ensembldb.ensembl.org
## Using cache in /homes/user/.vep/homo_sapiens/88_GRCh38
## Using API version 88, DB version 88
## polyphen version 2.2.2
## sift version sift5.2.2
## COSMIC version 78
## ESP version 20141103
## gencode version GENCODE 25
## genebuild version 2014-07
## HGMD-PUBLIC version 20162
## regbuild version 16
## assembly version GRCh38.p7
## ClinVar version 201610
## dbSNP version 147
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Feature_type : Type of feature - Transcript, RegulatoryFeature or MotifFeature
## Consequence : Consequence type
## cDNA_position : Relative position of base pair in cDNA sequence
## CDS_position : Relative position of base pair in coding sequence
## Protein_position : Relative position of amino acid in protein
## Amino_acids : Reference and variant amino acids
## Codons : Reference and variant codon sequence
## Existing_variation : Identifier(s) of co-located known variants
## Extra column keys:
## IMPACT : Subjective impact classification of consequence type
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
## FLAGS : Transcript quality flags
#Uploaded_variation  Location     Allele  Gene            Feature          Feature_type    Conse
rs7289170            22:17181903  G       ENSG00000093072 ENST00000262607  Transcript      synon
rs7289170            22:17181903  G       ENSG00000093072 ENST00000330232  Transcript      synon
```

The lines starting with "#" are header or meta information lines. The final one of these (highlighted in blue above) gives the column names for the data that follows. To see more information about VEP's output format, see the Data formats page.

We can see two lines of output here, both for the uploaded variant named rs7289170. In many cases, a variant will fall in more than one transcript. Typically this is where a single gene has multiple splicing variants. Here our variant has a consequence for the transcripts ENST00000262607 and ENST00000330232.

In the consequence column, we can see the consequence term synonymous_variant. This is terms forms part of an ontology for describing the effects of sequence variants on genomic features, produced by the Sequence Ontology (SO). See our predicted data page for a guide to the consequence types that VEP and Ensembl uses.

Let's try something a little more interesting. SIFT is an algorithm for predicting whether a given change in a protein sequence will be deleterious to the function of that protein. VEP can give SIFT predictions for most of the missense variants that it predicts. To do this, simply add --sift b (the b means we want **b**oth the prediction and the score):

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --sift b
```

SIFT calls variants either "deleterious" or "tolerated". We can use the VEP's filtering script to find only those that SIFT considers deleterious:

```
./filter_vep -i variant_effect_output.txt -filter "SIFT is deleterious" | grep -v "##" | head -n
#Uploaded_variation     Location        Allele  Gene            Feature         ...  Extra
rs2231495               22:17188416     C       ENSG00000093072 ENST00000262607 ...  SIFT=delete
rs2231495               22:17188416     C       ENSG00000093072 ENST00000399837 ...  SIFT=delete
rs2231495               22:17188416     C       ENSG00000093072 ENST00000399839 ...  SIFT=delete
rs115736959             22:19973143     A       ENSG00000099889 ENST00000263207 ...  SIFT=delete
```

Note that the SIFT score appears in the "Extra" column, as a key/value pair. This column can contain multiple key/value pairs depending on the options you give to VEP. See the Data formats page for more information on the fields in the Extra column.

You can also configure how VEP writes its output using the --fields flag.

You'll also see that we have multiple results for the same gene, ENSG00000093072. Let's say we're only interested in what is considered the canonical transcript for this gene (--canonical), and that we want to know what the commonly used gene symbol from HGNC is for this gene (--symbol). We can also use a UNIX pipe to pass the output from VEP directly into the filtering script:

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --sift b --canonical --symbo
--tab --fields Uploaded_variation,SYMBOL,CANONICAL,SIFT -o STDOUT | \
./filter_vep -filter "CANONICAL is YES and SIFT is deleterious"
...
#Uploaded_variation     SYMBOL  CANONICAL       SIFT
rs2231495               CECR1   YES     deleterious(0.05)
rs115736959             ARVCF   YES     deleterious(0.01)
rs116398106             ARVCF   YES     deleterious(0)
rs116782322             ARVCF   YES     deleterious(0)
...
rs115264708             PHF21B  YES     deleterious(0.03)
```

So now we can see all of the variants that have a deleterious effect on canonical transcripts, and the symbol for their genes. Nice!

For species with an Ensembl database of variants, VEP can annotate your input with identifiers and frequency data from variants co-located with your input data. For human, VEP's cache contains frequency data from 1000 Genomes, NHLBI-ESP and ExAC. Since our input file is from 1000 Genomes, let's add frequency data using --af_1kg:

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --af_1kg -o STDOUT | grep -v
#Uploaded_variation  Location     Allele  Gene             Feature          ...  Existing_variat
rs7289170            22:17181903  G       ENSG00000093072  ENST00000262607  ...  rs7289170
```

We can see frequency data for the AFR, AMR, EAS, EUR and SAS continental population groupings; these represent the frequency of the alternate (ALT) allele from our input (G in the case of rs7289170). Note that the Existing_variation column is populated by the identifier of the variant found in the VEP cache (and that it corresponds to the identifier from our input in Uploaded_variation). To retrieve only this information and not the frequency data, we could have used --check_existing (--af_1kg silently switches on --check_existing).

Over to you!

This has been just a short introduction to the capabilities of VEP - have a look through some more of the options, see them all on the command line using --help, or try using the shortcut --everything which switches on almost all available output fields! Try out the different options in the filtering script, and if you're feeling adventurous why not use some of your own data to annotate your variants or have a go with a plugin or two.

Variant Effect Predictor Download and install

Important notice to VEP users

- VEP is now available as the ensembl-vep package from GitHub
- The old version available as part of ensembl-tools will no longer be updated

Download

Use git to download the ensembl-vep package:

```
git clone https://github.com/Ensembl/ensembl-vep.git
cd ensembl-vep
```

Then follow the installation instructions.

Users without the git utility installed may download a zip file from GitHub, though we would always recommend using git if possible.

```
curl -L -O https://github.com/Ensembl/ensembl-vep/archive/release/88.zip
unzip 88.zip
cd ensembl-vep-release-88/
```

To update from a previous version:

```
cd ensembl-vep
git pull
git checkout release/88
perl INSTALL.pl
```

To use an older version (this example shows how to set up release 87):

```
cd ensembl-vep
git checkout release/87
perl INSTALL.pl
```

Previous versions (ensembl-tools)

Previously VEP was available as part of the ensembl-tools package (see the Ensembl archive site for documentation). The following downloads are available for archival purposes. [Show]

What's new

New in version 88 *(March 2017)*

- ensembl-vep is now the officially supported version of VEP
- Documentation updated to reflect switch to ensembl-vep. See the Ensembl archive site for documentation of the obsolete ensembl-tools VEP.
- The VEP script is now named simply `vep` (formerly `variant_effect_predictor.pl` or `vep.pl`)
- Directly use tabix-indexed GFF/GTF files as annotation sources
- Allele-specific reporting of frequencies (--af and more) and custom VCF annotations
- --check_existing now compares alleles by default, disable with --no_check_alleles
- Report the highest allele frequency observed in any population from 1000 genomes, ESP or ExAC using --max_af
- Get genomic HGVS nomenclature with --hgvsg
- Find the gene or transcript with the nearest transcription start site (TSS) to each input variant with --nearest
- filter_vep supports field/field comparisons e.g. AFR_AF > #EUR_AF
- Exclude predicted (XM and XR) transcripts when using RefSeq or merged cache with --exclude_predicted

- Filter transcripts used for annotation with --transcript_filter
- pileup input format no longer supported

Previous version history: Show

Requirements

VEP requires Perl (>=5.10 recommended, tested on 5.8, 5.10, 5.14, 5.18, 5.22) and the DBI and DBD::mysql package installed.

VEP's INSTALL.pl script will install required components of Ensembl API for you, but VEP may also be used with any pre-existing API installations you have **provided their versions match the version of VEP you are using**.

VEP has been developed for UNIX-like environments and works well on Linux (e.g. Ubuntu, Debian, Mint) and Mac OSX. It can also be used on Windows systems with a more involved installation process.

Installation

VEP's INSTALL.pl makes it easy to set up your environment for using the VEP. It will download and configure a minimal set of the Ensembl API for use by the VEP, and can also download cache files, FASTA files and plugins.

Run the following, and follow any prompts as they appear:

```
perl INSTALL.pl
```

Additional non-essential components and enhancements must be installed manually

Software components installed

- BioPerl
- ensembl
- ensembl-io
- ensembl-variation
- ensembl-funcgen
- Bio::DB::HTS

Users who already have the latest version of the API installed do not need to run the script, although may find it useful for getting an up-to-date API install (with post-release patches applied), and for retrieving cache and FASTA files. The API set installed by the script is local to the VEP, and will not affect any other Ensembl API installations.

The script will also attempt to install a Perl::XS module, Bio::DB::HTS, for rapid access to bgzipped FASTA files. If this fails, you may add the --NO_HTSLIB flag when running the installer; VEP will fall back to using Bio::DB::Fasta for this functionality (more details).

Running the installer

The installer script is run on the command line as follows:

```
 perl INSTALL.pl [options]
```

Users then follow on-screen prompts. Please heed any warnings, as when the script says it will delete/overwrite something, it really will!

Most users should not need to add any options, but configuration of the installer is possible with the following flags:

Flag	Alternate	Description
`--AUTO`	`-a`	Run installer without user prompts. Use "a" (API + Bio::DB::HTS/htslib), "l" (Bio::DB::HTS/htslib only), "c" (cache), "f" (FASTA), "p" (plugins) to specify parts to install e.g. -a ac for API and cache
`--SPECIES`	`-s`	Comma-separated list of species to install when using --AUTO. To install the RefSeq cache, add "_refseq" to the species name, e.g. "homo_sapiens_refseq", or "_merged" to install the merged Ensembl/RefSeq cache. Remember to use --refseq or --merged when running the VEP with the relevant cache!
`--ASSEMBLY`	`-y`	Assembly version to use when using --AUTO. Most species have only one assembly available on each software release; currently this is only required for human on release 76 onwards.
`--PLUGINS`	`-g`	Comma-separated list of plugins to install when using --AUTO. To install all available plugins,

		use "--PLUGINS all". To list available plugins, use "perl INSTALL.pl -a p --PLUGINS list".
`--VERSION [version]`	`-v`	By default the script will install the latest version of the Ensembl API (currently 88). Users can force the script to install a different version at their own risk
`--DESTDIR [dir]`	`-d`	By default the script will install the API modules in a subdirectory of the current directory named "Bio". Using this option users may configure where the Bio directory is created. If something other than the default is used, this directory must either be added to your PERL5LIB environment variable when running the VEP, or included using perl's -I flag: `perl -I [dir] vep`
`--CACHEDIR [dir]`	`-c`	By default the script will install the cache files in the ".vep" subdirectory of the user's home area. Using this option users can configure where cache files are installed. The --dir flag must be passed when running the VEP if a non-default directory is given: `./vep --dir [dir]`
`--UPDATE`	`-n`	Run the installer with this flag to check for and download new versions of the VEP. Any existing files are backed up. You will need to rerun the installer after update to retrieve update API, cache and FASTA files.
`--QUIET`	`-q`	Don't write any status output when using --AUTO.
`--PREFER_BIN`	`-p`	Use this if the installer fails with out of memory errors.
`--NO_HTSLIB`	`-l`	Don't attempt to install Bio::DB::HTS/htslib
`--NO_TEST`		Don't run API tests - useful if you know a harmless failure will prevent continuation of the installer

Additional components

INSTALL.pl will set up the minimum requirements for VEP, and for most users this will be adequate. Some features and enhancements, however, require the installation of additional components. Most are perl modules that are easily installed using cpanm; see this guide for more information on how to install perl modules.

- Additional features
  - JSON - required to produce JSON format output
  - Set::IntervalTree - used to find overlaps between entities in coordinate space. Required to use --nearest
  - Bio::DB::BigFile - required to use bigWig format custom annotation files. Requires the kent source tree for installation.
- Speed enhancements - these modules can improve VEP's runtime
  - PerlIO::gzip - marginal gains in compressed file parsing as used by VEP cache
  - ensembl-xs - provides pre-compiled replacements for frequently used routines in VEP. Requires manual installation, see README for details

Using VEP in Windows

VEP was developed as a command-line tool, and as a Perl script its natural environment is a Linux system. However, there are several ways you can use VEP on a Windows machine.

You may also consider using VEP's web or REST interfaces.

Virtual machines

Using a virtual machine you can run a virtual Linux system in a window on your machine. There are two ways to do this:

1. Use the Ensembl virtual machine image
2. Use Docker

DWIMperl

DWIMperl has a Windows package that contains base requirements for setting up VEP.

1. Download and install DWIMperl for Windows
2. Download and unpack the zip of the ensembl-vep package
3. Open a Command Prompt (search for Command Prompt in the Start Menu)
4. Navigate to the directory where you unpacked the VEP package, e.g.

   ```
   cd Downloads/ensembl-vep-release-87
   ```

5. Run INSTALL.pl with --NO_HTSLIB and --NO_TEST; you will see some warnings about the "which" command not being available (these will also appear when running VEP and can be ignored).

   ```
   perl INSTALL.pl --NO_HTSLIB --NO_TEST
   ```

Docker

Docker allows you to run applications in virtualised "containers". A docker image for VEP is available from DockerHub:

```
docker pull willmclaren/ensembl-vep
docker run -t -i willmclaren/ensembl-vep /bin/bash
./vep
```

Currently no volumes are pre-configured for the container; this is required if you wish to download data (e.g. cache files) that persists across sessions.

Variant Effect Predictor data formats

Input

Both the web and script version of VEP can use the same input formats. Formats can be auto-detected by the VEP script, but must be manually selected when using the web interface. VEP can use VCF, variant identifiers and HGVS notations in addition to the default format

Default

The default format is a simple **whitespace-separated** format (columns may be separated by space or tab characters), containing five required columns plus an optional identifier column:

1. **chromosome** - just the name or number, with no 'chr' prefix
2. **start**
3. **end**
4. **allele** - pair of alleles separated by a '/', with the reference allele first
5. **strand** - defined as + (forward) or - (reverse).
6. **identifier** - this identifier will be used in VEP's output. If not provided, VEP will construct an identifier from the given coordinates and alleles.

```
1   881907    881906    -/C   +
5   140532    140532    T/C   +
12  1017956   1017956   T/A   +
2   946507    946507    G/C   +
14  19584687  19584687  C/T   -
19  66520     66520     G/A   +    var1
8   150029    150029    A/T   +    var2
```

An insertion (of any size) is indicated by start coordinate = end coordinate + 1. For example, an insertion of 'C' between nucleotides 12600 and 12601 on the forward strand of chromosome 8 is indicated as follows:

```
8   12601     12600     -/C   +
```

A deletion is indicated by the exact nucleotide coordinates. For example, a three base pair deletion of nucleotides 12600, 12601, and 12602 of the reverse strand of chromosome 8 will be:

```
8   12600     12602     CGT/- -
```

VCF

VEP also supports using VCF (Variant Call Format) version 4.0. This is a common format used by the 1000 genomes project, and can be produced as an output format by many variant calling tools.

Users using VCF should note a peculiarity in the difference between how Ensembl and VCF describe unbalanced variants. For any unbalanced variant (i.e. insertion, deletion or unbalanced substitution), the VCF specification requires that the base immediately before the variant should be included in both the reference and variant alleles. This also affects the reported position i.e. the reported position will be one base before the actual site of the variant.

In order to parse this correctly, VEP needs to convert such variants into Ensembl-type coordinates, and it does this by removing the additional base and adjusting the coordinates accordingly. This means that if an identifier is not supplied for a variant (in the 3rd column of the VCF), then the identifier constructed and the position reported in VEP's output file will differ from the input.

This problem can be overcome by either:

1. ensuring each variant has a unique identifier specified in the 3rd column of the VCF
2. using VCF format as output (--vcf) - this preserves the formatting of your input coordinates and alleles

The following examples illustrate how VCF describes a variant and how it is handled internally by VEP. Consider the following aligned sequences (for the purposes of discussion on chromosome 20):

```
Ref: a t C g a // C is the reference base
1 : a t G g a // C base is a G in individual 1
```

```
2 : a t - g a // C base is deleted w.r.t. the reference in individual 2
3 : a t CAg a // A base is inserted w.r.t. the reference sequence in individual 3
```

Individual 1

The first individual shows a simple balanced substitution of G for C at base 3. This is described in a compatible manner in VCF and Ensembl styles. Firstly, in VCF:

```
20   3   .   C   G   .   PASS   .
```

And in Ensembl format:

```
 20   3   3   C/G   +
```

Individual 2

The second individual has the 3rd base deleted relative to the reference. In VCF, both the reference and variant allele columns must include the preceding base (T) and the reported position is that of the preceding base:

```
20   2   .   TC   T   .   PASS   .
```

In Ensembl format, the preceding base is not included, and the start/end coordinates represent the region of the sequence deleted. A "-" character is used to indicate that the base is deleted in the variant sequence:

```
20   3   3   C/-   +
```

The upshot of this is that while in the VCF input file the position of the variant is reported as 2, in the output file from VEP the position will be reported as 3. If no identifier is provided in the third column of the VCF, then the constructed identifier will be:

```
20_3_C/-
```

Individual 3

The third individual has an "A" inserted between the 3rd and 4th bases of the sequence relative to the reference. In VCF, as for the deletion, the base before the insertion is included in both the reference and variant allele columns, and the reported position is that of the preceding base:

```
20   3   .   C   CA   .   PASS   .
```

In Ensembl format, again the preceding base is not included, and the start/end positions are "swapped" to indicate that this is an insertion. Similarly to a deletion, a "-" is used to indicate no sequence in the reference:

```
 20   4   3   -/A   +
```

Again, the output will appear different, and the constructed identifier may not be what is expected:

```
20_3_-/A
```

The solution is to always add a unique identifer for each of your variants to the VCF file, or use VCF as your output format.

Structural variants

VEP can also call consequences on structural variants encoded in tab-delimited or VCF format. To recognise a variant as a structural variant, the allele string (or "SVTYPE" INFO field in VCF) must be set to one of the currently recognised values:

- **INS** - insertion
- **DEL** - deletion
- **DUP** - duplication
- **TDUP** - tandem duplication

Examples of structural variants encoded in tab-delimited format:

```
1    160283    471362    DUP
1    1385015   1387562   DEL
```

Examples of structural variants encoded in VCF format:

```
#CHROM   POS      ID   REF  ALT    QUAL  FILTER  INFO                     FORMAT
1        160283   sv1  .    <DUP>  .     .       SVTYPE=DUP;END=471362    .
1        1385015  sv2  .    <DEL>  .     .       SVTYPE=DEL;END=1387562   .
```

See the VCF definition document for more detail on how to describe structural variants in VCF format.

HGVS identifiers

See http://www.hgvs.org/mutnomen/ for details. These must be relative to genomic or Ensembl transcript coordinates. It also is possible to use RefSeq transcripts in both the web interface and the VEP script (see script documentation). This works for RefSeq transcripts that align to the genome correctly.

Examples:

```
ENST00000207771.3:c.344+626A>T
ENST00000471631.1:c.28_33delTCGCGG
ENST00000285667.3:c.1047_1048insC
5:g.140532T>C
```

Examples using RefSeq identifiers (using --refseq in the VEP script, or select the otherfeatures transcript database on the web interface and input type of HGVS):

```
NM_153681.2:c.7C>T
NM_005239.4:c.190G>A
NM_001025204.1:c.336G>A
```

HGVS protein notations may also be used, provided that they unambiguously map to a single genomic change. Due to redundancy in the amino acid code, it is not always possible to work out the corresponding genomic sequence change for a given protein sequence change. The following example is for a permissable protein notation in dog *(Canis familiaris)*:

```
ENSCAFP00000040171.1:p.Thr92Asn
```

HGVS notations may also be given in LRG coordinates:

```
LRG_1t1:c.841G>T
LRG_1:g.10006G>T
```

Variant identifiers

These should be e.g. dbSNP rsIDs, or any synonym for a variant present in the Ensembl Variation database. See here for a list of identifier sources in Ensembl.

Output

The default output format ("VEP" format when downloading from the web interface) is a 14 column tab-delimited file. Empty values are denoted by '-'. The output columns are:

1. **Uploaded variation** - as chromosome_start_alleles
2. **Location** - in standard coordinate format (chr:start or chr:start-end)
3. **Allele** - the variant allele used to calculate the consequence
4. **Gene** - Ensembl stable ID of affected gene
5. **Feature** - Ensembl stable ID of feature
6. **Feature type** - type of feature. Currently one of Transcript, RegulatoryFeature, MotifFeature.
7. **Consequence** - consequence type of this variant

8. **Position in cDNA** - relative position of base pair in cDNA sequence
9. **Position in CDS** - relative position of base pair in coding sequence
10. **Position in protein** - relative position of amino acid in protein
11. **Amino acid change** - only given if the variant affects the protein-coding sequence
12. **Codon change** - the alternative codons with the variant base in upper case
13. **Co-located variation** - known identifier of existing variant
14. **Extra** - this column contains extra information as key=value pairs separated by ";", see below.

Other output fields

- **IMPACT** - the impact modifier for the consequence type
- **VARIANT_CLASS** - Sequence Ontology variant class
- **SYMBOL** - the gene symbol
- **SYMBOL_SOURCE** - the source of the gene symbol
- **STRAND** - the DNA strand (1 or -1) on which the transcript/feature lies
- **ENSP** - the Ensembl protein identifier of the affected transcript
- **FLAGS** - transcript quality flags:
  - *cds_start_NF:* CDS 5' incomplete
  - *cds_end_NF:* CDS 3' incomplete
- **SWISSPROT** - Best match UniProtKB/Swiss-Prot accession of protein product
- **TREMBL** - Best match UniProtKB/TrEMBL accession of protein product
- **UNIPARC** - Best match UniParc accession of protein product
- **HGVSc** - the HGVS coding sequence name
- **HGVSp** - the HGVS protein sequence name
- **HGVSg** - the HGVS genomic sequence name
- **HGVS_OFFSET** - Indicates by how many bases the HGVS notations for this variant have been shifted
- **NEAREST** - Identifier(s) of nearest transcription start site
- **SIFT** - the SIFT prediction and/or score, with both given as prediction(score)
- **PolyPhen** - the PolyPhen prediction and/or score
- **MOTIF_NAME** - the source and identifier of a transcription factor binding profile aligned at this position
- **MOTIF_POS** - The relative position of the variation in the aligned TFBP
- **HIGH_INF_POS** - a flag indicating if the variant falls in a high information position of a transcription factor binding profile (TFBP)
- **MOTIF_SCORE_CHANGE** - The difference in motif score of the reference and variant sequences for the TFBP
- **CELL_TYPE** - List of cell types and classifications for regulatory feature
- **CANONICAL** - a flag indicating if the transcript is denoted as the canonical transcript for this gene
- **CCDS** - the CCDS identifer for this transcript, where applicable
- **INTRON** - the intron number (out of total number)
- **EXON** - the exon number (out of total number)
- **DOMAINS** - the source and identifer of any overlapping protein domains
- **DISTANCE** - Shortest distance from variant to transcript
- **IND** - individual name
- **ZYG** - zygosity of individual genotype at this locus
- **SV** - IDs of overlapping structural variants
- **FREQS** - Frequencies of overlapping variants used in filtering
- **AF** - Frequency of existing variant in 1000 Genomes
- **AFR_AF** - Frequency of existing variant in 1000 Genomes combined African population
- **AMR_AF** - Frequency of existing variant in 1000 Genomes combined American population
- **ASN_AF** - Frequency of existing variant in 1000 Genomes combined Asian population

- **EUR_AF** - Frequency of existing variant in 1000 Genomes combined European population
- **EAS_AF** - Frequency of existing variant in 1000 Genomes combined East Asian population
- **SAS_AF** - Frequency of existing variant in 1000 Genomes combined South Asian population
- **AA_AF** - Frequency of existing variant in NHLBI-ESP African American population
- **EA_AF** - Frequency of existing variant in NHLBI-ESP European American population
- **ExAC_AF** - Frequency of existing variant in ExAC combined population
- **ExAC_Adj_AF** - Adjusted frequency of existing variant in ExAC combined population
- **ExAC_AFR_AF** - Frequency of existing variant in ExAC African/American population
- **ExAC_AMR_AF** - Frequency of existing variant in ExAC American population
- **ExAC_EAS_AF** - Frequency of existing variant in ExAC East Asian population
- **ExAC_FIN_AF** - Frequency of existing variant in ExAC Finnish population
- **ExAC_NFE_AF** - Frequency of existing variant in ExAC Non-Finnish European population
- **ExAC_OTH_AF** - Frequency of existing variant in ExAC combined other combined populations
- **ExAC_SAS_AF** - Frequency of existing variant in ExAC South Asian population
- **MAX_AF** - Maximum observed allele frequency in 1000 Genomes, ESP and ExAC
- **MAX_AF_POPS** - Populations in which maximum allele frequency was observed
- **CLIN_SIG** - ClinVar clinical significance of the dbSNP variant
- **BIOTYPE** - Biotype of transcript or regulatory feature
- **APPRIS** - Annotates alternatively spliced transcripts as primary or alternate based on a range of computational methods. NB: not available for GRCh37
- **TSL** - Transcript support level. NB: not available for GRCh37
- **PUBMED** - Pubmed ID(s) of publications that cite existing variant
- **SOMATIC** - Somatic status of existing variant(s)
- **PHENO** - Indicates if existing variant is associated with a phenotype, disease or trait
- **GENE_PHENO** - Indicates if overlapped gene is associated with a phenotype, disease or trait
- **ALLELE_NUM** - Allele number from input; 0 is reference, 1 is first alternate etc
- **MINIMISED** - Alleles in this variant have been converted to minimal representation before consequence calculation
- **PICK** - indicates if this block of consequence data was picked by --flag_pick or --flag_pick_allele
- **REFSEQ_MATCH** - the RefSeq transcript match status; contains a number of flags indicating whether this RefSeq transcript matches the underlying reference sequence and/or an Ensembl transcript (more information). NB: not available for GRCh37.
  - *rseq_3p_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the 3' UTR of the RefSeq model with respect to the primary genome assembly (e.g. GRCh37/GRCh38).
  - *rseq_5p_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the 5' UTR of the RefSeq model with respect to the primary genome assembly.
  - *rseq_cds_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the CDS of the RefSeq model with respect to the primary genome assembly.
  - *rseq_ens_match_cds:* signifies that for the RefSeq transcript there is an overlapping Ensembl model that is identical across the CDS region only. A CDS match is defined as follows: the CDS and peptide sequences are identical and the genomic coordinates of every translatable exon match. Useful related attributes are: rseq_ens_match_wt and rseq_ens_no_match.
  - *rseq_ens_match_wt:* signifies that for the RefSeq transcript there is an overlapping Ensembl model that is identical across the whole transcript. A whole transcript match is defined as follows: 1) In the case that both models are coding, the transcript, CDS and peptide sequences are all identical and the genomic coordinates of every exon match. 2) In the case that both transcripts are non-coding the transcript sequences and the genomic coordinates of every exon are identical. No comparison is made between a coding and a non-coding transcript. Useful related attributes are: rseq_ens_match_cds and rseq_ens_no_match.
  - *rseq_ens_no_match:* signifies that for the RefSeq transcript there is no overlapping Ensembl model that is identical across either the whole transcript or the CDS. This is caused by differences between the transcript, CDS or peptide sequences or between the exon genomic coordinates. Useful related attributes are: rseq_ens_match_wt and rseq_ens_match_cds.
  - *rseq_mrna_match:* signifies an exact match between the RefSeq transcript and the underlying primary genome assembly sequence (based on a match between the transcript stable id and an accession in the RefSeq mRNA file). An exact match occurs when the underlying genomic sequence of the model can be perfectly aligned to the mRNA sequence post polyA clipping.

- *rseq_mrna_nonmatch:* signifies a non-match between the RefSeq transcript and the underlying primary genome assembly sequence. A non-match is deemed to have occurred if the underlying genomic sequence does not have a perfect alignment to the mRNA sequence post polyA clipping. It can also signify that no comparison was possible as the model stable id may not have had a corresponding entry in the RefSeq mRNA file (sometimes happens when accessions are retired or changed). When a non-match occurs one or several of the following transcript attributes will also be present to provide more detail on the nature of the non-match: rseq_5p_mismatch, rseq_cds_mismatch, rseq_3p_mismatch, rseq_nctran_mismatch, rseq_no_comparison
- *rseq_nctran_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. This is a comparison between the entire underlying genomic sequence of the RefSeq model to the mRNA in the case of RefSeq models that are non-coding.
- *rseq_no_comparison:* signifies that no alignment was carried out between the underlying primary genome assembly sequence and a corresponding RefSeq mRNA. The reason for this is generally that no corresponding, unversioned accession was found in the RefSeq mRNA file for the transcript stable id. This sometimes happens when accessions are retired or replaced. A second possibility is that the sequences were too long and problematic to align (though this is rare).

```
11_224088_C/A    11:224088   A  ENSG00000142082  ENST00000525319  Transcript         missense_variar
11_224088_C/A    11:224088   A  ENSG00000142082  ENST00000534381  Transcript         5_prime_UTR_var
11_224088_C/A    11:224088   A  ENSG00000142082  ENST00000529055  Transcript         downstream_vari
11_224585_G/A    11:224585   A  ENSG00000142082  ENST00000529937  Transcript         intron_variant
22_16084370_G/A  22:16084370 A  -                ENSR00000615113  RegulatoryFeature  regulatory_regi
```

The VEP script will also add a header to the output file. This contains information about the databases connected to, and also a key describing the key/value pairs used in the extra column.

```
## ENSEMBL VARIANT EFFECT PREDICTOR v88.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_88_38 on ensembldb.ensembl.org
## Using cache in /homes/user/.vep/homo_sapiens/88_GRCh38
## Using API version 88, DB version 88
## polyphen version 2.2.2
## sift version sift5.2.2
## COSMIC version 78
## ESP version 20141103
## gencode version GENCODE 25
## genebuild version 2014-07
## HGMD-PUBLIC version 20162
## regbuild version 16
## assembly version GRCh38.p7
## ClinVar version 201610
## dbSNP version 147
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Feature_type : Type of feature - Transcript, RegulatoryFeature or MotifFeature
## Consequence : Consequence type
## cDNA_position : Relative position of base pair in cDNA sequence
## CDS_position : Relative position of base pair in coding sequence
## Protein_position : Relative position of amino acid in protein
## Amino_acids : Reference and variant amino acids
## Codons : Reference and variant codon sequence
## Existing_variation : Identifier(s) of co-located known variants
## Extra column keys:
## IMPACT : Subjective impact classification of consequence type
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
## FLAGS : Transcript quality flags
```

Tab-delimited output

The --tab flag instructs VEP to write output as a tab-delimited table. This differs from the default output format in that each individual field from the "Extra" field is written to a separate tab-delimited column. This makes the output more suitable for import into spreadsheet programs such as Excel. This is also the format used when selecting the "TXT" option on the VEP web interface.

The choice and order of columns in the output may be configured using --fields.

VCF output

The VEP script can also generate VCF output using the --vcf flag. Consequences are added in the INFO field of the VCF file, using the key "CSQ" (configure this using --vcf_info_field). Data fields are encoded separated by "|"; the order of fields is written in the VCF header. Output fields can be configured by using --fields. Unpopulated fields are represented by an empty string.

VCFs produced by VEP can be filtered by filter_vep.pl in the same way as standard format output files.

If the input format was VCF, the file will remain unchanged save for the addition of the CSQ field and the header (unless using any filtering). If an existing CSQ field is found, it will be replaced by the one added by the VEP (use --keep_csq to preserve it).

Custom data added with --custom are added as separate fields, using the key specified for each data file.

Commas in fields are replaced with ampersands (&) to preserve VCF format.

```
##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from Ensembl VEP. Format: A
#CHROM   POS        ID           REF   ALT   QUAL   FILTER   INFO
21       26978790   rs75377686   T     C     .      .        CSQ=C|missense_variant|MODERATE|MRPL39|ENSG000
```

JSON output

VEP can produce output in the form of serialised JSON objects using the --json flag. JSON is a serialisation format that can be parsed and processed easily by many packages and programming languages; it is used as the default output format for Ensembl's REST server.

Each input variant is reported as a single JSON object which constitutes one line of the output file. The JSON object is structured somewhat differently to the other VEP output formats, in that per-variant fields (e.g. co-located existing variant details) are reported only once. Consequences are grouped under the feature type that they affect (Transcript, Regulatory Feature, etc). The original input line (e.g. from VCF input) is reported under the "input" key in order to aid aligning input with output.

Here follows an example of JSON output (prettified and redacted for display here):

```
{
  "input": "1 230845794 test1 A G . . .",
  "id": "test1",
  "seq_region_name": "1",
  "start": 230845794,
  "end": 230845794,
  "strand": 1,
  "allele_string": "A/G",
  "most_severe_consequence": "missense_variant",
  "colocated_variants": [
    {
      "id": "rs699",
      "seq_region_name": "1",
      "start": 230845794,
      "end": 230845794,
      "strand": 1,
      "allele_string": "A/G",
      "minor_allele": "A",
      "minor_allele_freq": 0.3384,
      "afr_allele": "A",
      "afr_maf": 0.13,
      "amr_allele": "A",
      "amr_maf": 0.36,
      "asn_allele": "A",
      "asn_maf": 0.16,
      "eur_allele": "A",
      "eur_maf": 0.41,
      "pubmed": [
        18513389,
        23716723
      ]
    },
    {
      "seq_region_name": "1",
      "strand": 1,
      "id": "COSM425562",
      "allele_string": "A/G",
      "start": 230845794,
      "end": 230845794
    }
  ],
  "transcript_consequences": [
    {
      "variant_allele": "G",
      "consequence_terms": [
        "missense_variant"
      ],
      "gene_id": "ENSG00000135744",
      "gene_symbol": "AGT",
      "gene_symbol_source": "HGNC",
      "transcript_id": "ENST00000366667",
      "biotype": "protein_coding",
      "strand": -1,
      "cdna_start": 1018,
      "cdna_end": 1018,
      "cds_start": 803,
      "cds_end": 803,
      "protein_start": 268,
      "protein_end": 268,
      "codons": "aTg/aCg",
      "amino_acids": "M/T",
      "polyphen_prediction": "benign",
      "polyphen_score": 0,
      "sift_prediction": "tolerated",
      "sift_score": 1,
      "hgvsc": "ENST00000366667.4:c.803T>C",
      "hgvsp": "ENSP00000355627.4:p.Met268Thr"
    }
  ],
  "regulatory_feature_consequences": [
    {
      "variant_allele": "G",
      "consequence_terms": [
        "regulatory_region_variant"
      ],
      "regulatory_feature_id": "ENSR00001529861"
    }
  ]
}
```

In accordance with JSON conventions, all keys are lower-case. Some keys also have different names and structures to those found in the other VEP output formats:

Key	JSON equivalent(s)	Notes
Consequence	consequence_terms	
Gene	gene_id	
Feature	transcript_id, regulatory_feature_id, motif_feature_id	Consequences are grouped under the feature type they affect
ALLELE	variant_allele	
SYMBOL	gene_symbol	
SYMBOL_SOURCE	gene_symbol_source	
ENSP	protein_id	
OverlapBP	bp_overlap	
OverlapPC	percentage_overlap	
Uploaded_variation	id	
Location	seq_region_name,	The variant's location field is broken down into constituent coordinate parts for clarity.

	start, end, strand	"seq_region_name" is used in place of "chr" or "chromosome" for consistency with other parts of Ensembl's REST API
GMAF	minor_allele, minor_allele_freq	
*_maf	*_allele, *_maf	
cDNA_position	cdna_start, cdna_end	
CDS_position	cds_start, cds_end	
Protein_position	protein_start, protein_end	
SIFT	sift_prediction, sift_score	
PolyPhen	polyphen_prediction, polyphen_score	

Statistics

VEP writes an HTML file containing statistics pertaining to the results of your job; it is named **[output_file]_summary.html** (with the default options the file will be named **variant_effect_output.txt_summary.html**). To view it you should open the file in your web browser.

To prevent VEP writing a stats file, use the flag --no_stats. To have VEP write a machine-readable text file in place of the HTML, use --stats_text. To change the name of the stats file from the default, use --stats_file [file].

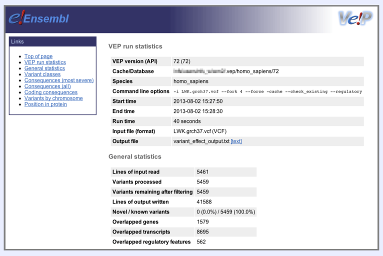
The page contains several sections:

General statistics

This section contains two tables. The first describes the cache and/or database used, the version of VEP, species, command line parameters, input/output files and run time. The second table contains information about the number of variants, and the number of genes, transcripts and regulatory features overlapped by the input.

Charts and tables

There then follows several charts, most with accompanying tables. Tables and charts are interactive; clicking on a row to highlight it in the table will highlight the relevant segment in the chart, and vice versa.



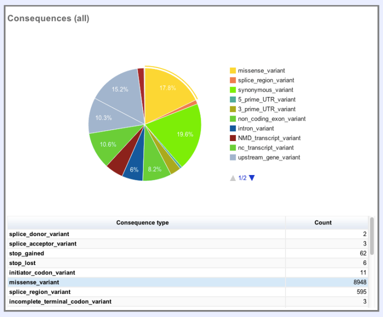
VEP run statistics

VEP version (API)	72 (72)
Cache/Database	[redacted].vep/homo_sapiens/72
Species	homo_sapiens
Command line options	`-i LWK.grch37.vcf --fork 4 --force -cache --check_existing --regulatory`
Start time	2013-08-02 15:27:50
End time	2013-08-02 15:28:30
Run time	40 seconds
Input file (format)	LWK.grch37.vcf (VCF)
Output file	variant_effect_output.txt [text]

General statistics

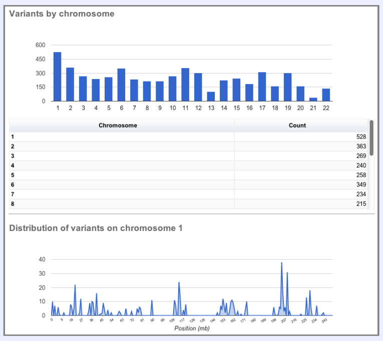
Lines of input read	5461
Variants processed	5459
Variants remaining after filtering	5459
Lines of output written	41588
Novel / known variants	0 (0.0%) / 5459 (100.0%)
Overlapped genes	1579
Overlapped transcripts	8695
Overlapped regulatory features	562

General statistics



Consequence type	Count
splice_donor_variant	2
splice_acceptor_variant	3
stop_gained	62
stop_lost	6
initiator_codon_variant	11
missense_variant	8948
splice_region_variant	595
incomplete_terminal_codon_variant	3

Summary of called consequence types



Chromosome	Count
1	528
2	363
3	269
4	240
5	258
6	349
7	234
8	215

Distribution of variants across chromosomes

VEP is run on the command line as follows (assuming you are in the ensembl-vep directory):

```
./vep [options]
```

where [options] represent a set of flags and options to the script. A basic set of flags can be listed using --help:

```
./vep --help
```

Users should download a cache file for their species of interest, using either the installer script or by following the documentation, and run VEP with either the --cache or --offline option.

For smaller input files, it is possible for VEP to connect to Ensembl's public database servers in place of the cache; to enable this, use --database

Most users will need to use only a few of the options described below; for most the following command will be enough to get started with:

```
./vep --cache -i input.txt -o output.txt
```

where input.txt contains data in one of the compatible input formats, and output.txt is the output file created by the script. See Data Formats for more detail on input and output formats.

Options can be passed as the full string (e.g. --format), or as the shortest unique string among the options (e.g. --form for --format, since there is another option --force_overwrite).

You may use one or two hypen ("-") characters before each option name; **`--cache`** or **`-cache`**.

Options can also be read from a configuration file - either passively stored as $HOME/.vep/vep.ini, or actively using --config.

Basic options

Flag	Alternate	Description
`--help`		Display help message and quit
`--quiet`	`-q`	Suppress warning messages. *Not used by default*

`--config [filename]`		Load configuration options from a config file. The config file should consist of whitespace-separated pairs of option names and settings e.g.:
		`output_file    my_output.txt`
		`species        mus_musculus`
		`format         vcf`
		`host           useastdb.ensembl.org`
		
		A config file can also be implicitly read; save the file as $HOME/.vep/vep.ini (or equivalent directory if using --dir). Any options in this file will be overridden by those specified in a config file using --config, and in turn by any options manually specified on the command line. You can create a quick version file of this by setting the flags as normal and running the script in verbose (**-v**) mode. This will output lines that can be copied to a config file that can be loaded in on the next run using --config. *Not used by default*

`--everything`		Shortcut flag to switch on all of the following: --sift b, --polyphen b, --ccds, --uniprot, --hgvs, --symbol, --numbers, --domains, --regulatory, --canonical, --protein, --biotype, --uniprot, --tsl, --appris, --gene_phenotype --af, --af_1kg, --af_esp, --af_exac, --max_af, --pubmed, --variant_class
`--fork [num_forks]`		Enable forking, using the specified number of forks. Forking can dramatically improve the runtime of the script. *Not used by default*

Input options

Flag	Alternate	Description
`--species [species]`		Species for your data. This can be the latin name e.g. "homo_sapiens" or any Ensembl alias e.g. "mouse". Specifying the latin name can speed up initial database connection as the registry does not have to load all available database aliases on the server. *Default = "homo_sapiens"*
`--assembly [name]`	`-i`	Select the assembly version to use if more than one available. If using the cache, you must have the appropriate assembly's cache file installed. If not specified and you have only 1 assembly version installed, this will be chosen by default. *Default = use found assembly version*
`--input_file [filename]`	`-i`	Input file name. If not specified, the script will attempt to read from STDIN.
`--format [format]`		Input file format - one of "ensembl", "vcf", "hgvs", "id". By default, the script auto-detects the input file format. Using this option you can force the script to read the input file as Ensembl, VCF, IDs or HGVS. *Auto-detects format by default*
`--output_file [filename]`	`-o`	Output file name. The script can write to STDOUT by specifying STDOUT as the output file name - this will force quiet mode. *Default = "variant_effect_output.txt"*
`--force_overwrite`	`--force`	By default, the script will fail with an error if the output file already exists. You can force the overwrite of the existing file by using this flag. *Not used by default*
`--stats_file [filename]`		Summary stats file name. This is an HTML file containing a summary of the VEP run - the file name must end ".htm" or ".html". *Default = "variant_effect_output.txt_summary.html"*
`--no_stats`		Don't generate a stats file. Provides marginal gains in run time.
`--stats_text`		Generate a plain text stats file in place of the HTML.

Cache options

Flag	Alternate	Description
`--cache`		Enables use of the cache. Add --refseq or --merged to use the refseq or merged cache, (if installed).
`--dir [directory]`		Specify the base cache/plugin directory to use. *Default = "$HOME/.vep/"*
`--dir_cache [directory]`		Specify the cache directory to use. *Default = "$HOME/.vep/"*
`--dir_plugins [directory]`		Specify the plugin directory to use. *Default = "$HOME/.vep/"*
`--offline`		Enable offline mode. No database connections will be made, and a cache file or GFF/GTF file is required for annotation. Add --refseq to use the refseq cache (if installed). *Not used by default*
`--fasta [file|dir]`		Specify a FASTA file or a directory containing FASTA files to use to look up reference sequence. The first time you run the script with this parameter an index will be built which can take a few minutes. This is required if fetching HGVS annotations (--hgvs) or checking reference sequences (--check_ref) in offline mode (--offline), and optional with some performance increase in cache mode (--cache). See documentation for more details. *Not used by default*
`--refseq`		Specify this option if you have installed the RefSeq cache in order for VEP to pick up the alternate cache directory. This cache contains transcript objects corresponding to RefSeq transcripts (to include CCDS and Ensembl ESTs also, use --all_refseq). Consequence output will be given relative to these transcripts in place of the default Ensembl transcripts (see documentation)
`--merged`		Use the merged Ensembl and RefSeq cache. Consequences are flagged with the SOURCE of each transcript used.
`--cache_version`		Use a different cache version than the assumed default (the VEP version). This should be used with Ensembl Genomes caches since their version numbers do not match Ensembl versions. For example, the VEP/Ensembl version may be 88 and the Ensembl Genomes version 35. *Not used by default*
`--show_cache_info`		Show source version information for selected cache and quit

`--buffer_size [number]`		Sets the internal buffer size, corresponding to the number of variants that are read in to memory simultaneously. Set this lower to use less memory at the expense of longer run time, and higher to use more memory with a faster run time. *Default = 5000*

Other annotation sources

Flag	Alternate	Description
`--plugin [plugin name]`		Use named plugin. Plugin modules should be installed in the Plugins subdirectory of the VEP cache directory (defaults to $HOME/.vep/). Multiple plugins can be used by supplying the --plugin flag multiple times. See plugin documentation. *Not used by default*
`--custom [filename]`		Add custom annotation to the output. Files must be tabix indexed or in the bigWig format. Multiple files can be specified by supplying the --custom flag multiple times. See here for full details. *Not used by default*
`--gff [filename]`		Use GFF transcript annotations in [filename] as an annotation source. Requires a FASTA file of genomic sequence.*Not used by default*
`--gtf [filename]`		Use GTF transcript annotations in [filename] as an annotation source. Requires a FASTA file of genomic sequence.*Not used by default*
`--bam [filename]`		**ADVANCED** Use BAM file of sequence alignments to correct transcript models not derived from reference genome sequence. Used to correct RefSeq transcript models *Not used by default*

Output options

Flag	Alternate	Description
`--variant_class`		Output the Sequence Ontology variant class. *Not used by default*
`--sift [p|s|b]`		Species limited SIFT predicts whether an amino acid substitution affects protein function based on sequence homology and the physical properties of amino acids. VEP can output the **p**rediction term, **s**core or **b**oth. *Not used by default*
`--polyphen [p|s|b]`	`--poly`	Human only PolyPhen is a tool which predicts possible impact of an amino acid substitution on the structure and function of a human protein using straightforward physical and comparative considerations. VEP can output the **p**rediction term, **s**core or **b**oth. VEP uses the humVar score by default - use --humdiv to retrieve the humDiv score. *Not used by default*
`--nearest [transcript|gene|symbol]`		Retrieve the transcript or gene with the nearest protein-coding transcription start site (TSS) to each input variant. Use "transcript" to retrieve the transcript stable ID, "gene" to retrieve the gene stable ID, or "symbol" to retrieve the gene symbol. Note that the nearest TSS may not belong to a transcript that overlaps the input variant, and more than one may be reported in the case where two are equidistant from the input coordinates. Currently only available when using a cache annotation source, and requires the Set::IntervalTree perl module. *Not used by default*
`--humdiv`		Human only Retrieve the humDiv PolyPhen prediction instead of the default humVar. *Not used by default*
`--gene_phenotype`		Indicates if the overlapped gene is associated with a phenotype, disease or trait. See list of phenotype sources. *Not used by default*
`--regulatory`		Look for overlaps with regulatory regions. The script can also call if a variant falls in a high information position within a transcription factor binding site. Output lines have a Feature type of RegulatoryFeature or MotifFeature. *Not used by default*
`--cell_type`		Report only regulatory regions that are found in the given cell type(s). Can be a single cell type or a comma-separated list. The functional type in each cell type is reported under CELL_TYPE in the output. To retrieve a list of cell types, use --cell_type list. *Not used by default*
`--individual [all|ind list]`		Consider only alternate alleles present in the genotypes of the specified

		individual(s). May be a single individual, a comma-separated list or "all" to assess all individuals separately. Individual variant combinations homozygous for the given reference allele will not be reported. Each individual and variant combination is given on a separate line of output. Only works with VCF files containing individual genotype data; individual IDs are taken from column headers. *Not used by default*
`--phased`		Force VCF genotypes to be interpreted as phased. For use with plugins that depend on phased data. *Not used by default*
`--allele_number`		Identify allele number from VCF input, where 1 = first ALT allele, 2 = second ALT allele etc. Useful when using --minimal *Not used by default*
`--total_length`		Give cDNA, CDS and protein positions as Position/Length. *Not used by default*
`--numbers`		Adds affected exon and intron numbering to to output. Format is Number/Total. *Not used by default*
`--domains`		Adds names of overlapping protein domains to output. *Not used by default*
`--no_escape`		Don't URI escape HGVS strings. *Default = escape*
`--keep_csq`		Don't overwrite existing CSQ entry in VCF INFO field. *Overwrites by default*
`--vcf_info_field [CSQ|ANN|(other)]`		Change the name of the INFO key that VEP write the consequences to in its VCF output. Use "ANN" for compatibility with other tools such as snpEff. *Default: CSQ*
`--terms [ensembl|so]`	`-t`	The type of consequence terms to output. The Ensembl terms are described here. The Sequence Ontology is a joint effort by genome annotation centres to standardise descriptions of biological sequences. *Default = "SO"*

Identifiers

Flag	Alternate	Description
`--hgvs`		Add HGVS nomenclature based on Ensembl stable identifiers to the output. Both coding and protein sequence names are added where appropriate. To generate HGVS identifiers when using --cache or --offline you must use a FASTA file and --fasta. HGVS notations given on Ensembl identifiers are versioned. *Not used by default*
`--hgvsg`		Add genomic HGVS nomenclature based on the input chromosome name. To generate HGVS identifiers when using --cache or --offline you must use a FASTA file and --fasta. *Not used by default*
`--shift_hgvs [0|1]`		Enable or disable 3' shifting of HGVS notations. When enabled, this causes ambiguous insertions or deletions (typically in repetetive sequence tracts) to be "shifted" to their most 3' possible coordinates (relative to the transcript sequence and strand) before the HGVS notations are calculated; the flag HGVS_OFFSET is set to the number of bases by which the variant has shifted, relative to the input genomic coordinates. Disabling retains the original input coordinates of the variant. *Default: 1 (shift)*
`--protein`		Add the Ensembl protein identifier to the output where appropriate. *Not used by default*
`--symbol`		Adds the gene symbol (e.g. HGNC) (where available) to the output. *Not used by default*
`--ccds`		Adds the CCDS transcript identifer (where available) to the output. *Not used by default*
`--uniprot`		Adds best match accessions for translated protein products from three UniProt-related databases (SWISSPROT, TREMBL and UniParc) to the output. *Not used by default*
`--tsl`		Adds the transcript support level for this transcript to the output. NB: not available for GRCh37.*Not used by default*
`--appris`		Adds the APPRIS isoform annotation for this transcript to the output. NB: not available for GRCh37.*Not used by default*
`--canonical`		Adds a flag indicating if the transcript is the canonical transcript for the gene. *Not used by default*
`--biotype`		Adds the biotype of the transcript or regulatory feature. *Not used by default*

Flag	Alternate	Description
`--xref_refseq`		Output aligned RefSeq mRNA identifier for transcript. NB: theRefSeq and Ensembl transcripts aligned in this way MAY NOT, AND FREQUENTLY WILL NOT, match exactly in sequence, exon structure and protein product. *Not used by default*
`--synonyms [file]`		Load a file of chromosome synonyms. File should be tab-delimited with the primary identifier in column 1 and the synonym in column 2. Synonyms are used bi-directionally so columns may be switched. Synoyms allow different chromosome identifiers to be used in the input file and any annotation source (cache, database, GFF, custom file, FASTA file). *Not used by default*

Co-located variants

Flag	Alternate	Description
`--check_existing`		Checks for the existence of known variants that are co-located with your input. By default the alleles are compared - to compare only coordinates, use --no_check_alleles. *Not used by default*
`--no_check_alleles`		When checking for existing variants, by default VEP only reports a co-located variant if none of the input alleles are novel. For example, if the user input has alleles A/G, and an existing co-located variant has alleles A/C, the co-located variant will not be reported. Strand is also taken into account - in the same example, if the user input has alleles T/G but on the negative strand, then the co-located variant **will** be reported since its alleles match the reverse complement of user input. Use this flag to disable this behaviour and compare using coordinates alone. *Not used by default*
`--af`		Add the global allele frequency (AF) from 1000 Genomes Phase 3 data for any known co-located variant to the output. For this and all --af_* flags, the frequency reported is for the **input allele** only, not necessarily the non-reference or derived allele *Not used by default*
`--max_af`		Report the highest allele frequency observed in any population from 1000 genomes, ESP or ExAC. *Not used by default*
`--af_1kg`		Add allele frequency from continental populations (AFR,AMR,EAS,EUR,SAS) of 1000 Genomes Phase 3 to the output. Must be used with --cache *Not used by default*
`--af_esp`		Include allele frequency from NHLBI-ESP populations. Must be used with --cache *Not used by default*
`--af_exac`		Include allele frequency from ExAC project populations. Must be used with --cache *Not used by default*
`--pubmed`		Report Pubmed IDs for publications that cite existing variant. Must be used with --cache. *Not used by default*
`--failed [0|1]`		When checking for co-located variants, by default the script will exclude variants that have been flagged as failed. Set this flag to include such variants. *Default: 0 (exclude)*

Data format options

Flag	Alternate	Description
`--vcf`		Writes output in VCF format. Consequences are added in the INFO field of the VCF file, using the key "CSQ". Data fields are encoded separated by "|"; the order of fields is written in the VCF header. Output fields can be selected by using --fields. If the input format was VCF, the file will remain unchanged save for the addition of the CSQ field (unless using any filtering). Custom data added with --custom are added as separate fields, using the key specified for each data file. Commas in fields are replaced with ampersands (&) to preserve VCF format. *Not used by default*
`--tab`		Writes output in tab-delimited format. *Not used by default*
`--json`		Writes output in JSON format. *Not used by default*

`--fields [list]`	Configure the output format using a comma separated list of fields. Fields may be those present in the default output columns, or any of those that appear in the Extra column (including those added by plugins or custom annotations). Output remains tab-delimited. Can only be used with tab or VCF format output. *Not used by default*
`--minimal`	Convert alleles to their most minimal representation before consequence calculation i.e. sequence that is identical between each pair of reference and alternate alleles is trimmed off from both ends, with coordinates adjusted accordingly. Note this may lead to discrepancies between input coordinates and coordinates reported by VEP relative to transcript sequences; to avoid issues, use --allele_number and/or ensure that your input variants have unique identifiers. The MINIMISED flag is set in the VEP output where relevant. *Not used by default*

Filtering and QC options

NOTE: The filtering options here filter your results **before** they are written to your output file. Using VEP's filtering script, it is possible to filter your results **after** VEP has run. This way you can retain all of the results and run multiple filter sets on the same results to find different data of interest.

Flag	Alternate	Description
`--gencode_basic`		Limit your analysis to transcripts belonging to the GENCODE basic set. This set has fragmented or problematic transcripts removed. *Not used by default*
`--all_refseq`		When using the RefSeq or merged cache, include e.g. CCDS and Ensembl EST transcripts in addition to those from RefSeq (see documentation). Only works when using --refseq or --merged
`--exclude_predicted`		When using the RefSeq or merged cache, exclude predicted transcripts (i.e. those with identifiers beginning with "XM_" or "XR_").
`--transcript_filter`		**ADVANCED** Filter transcripts according to any arbitrary set of rules. Uses similar notation to filter_vep. You may filter on any key defined in the root of the transcript object; most commonly this will be "stable_id": `--transcript_filter "stable_id match N[MR]_"`
`--check_ref`		Force the script to check the supplied reference allele against the sequence stored in the Ensembl Core database or supplied FASTA file. Lines that do not match are skipped. *Not used by default*
`--dont_skip`		Don't skip input variants that fail validation, e.g. those that fall on unrecognised sequences
`--allow_non_variant`		When using VCF format as input and output, by default VEP will skip non-variant lines of input (where the ALT allele is null). Enabling this option the lines will be printed in the VCF output with no consequence data added.
`--chr [list]`		Select a subset of chromosomes to analyse from your file. Any data not on this chromosome in the input will be skipped. The list can be comma separated, with "-" characters representing an interval. For example, to include chromosomes 1, 2, 3, 10 and X you could use --chr 1-3,10,X *Not used by default*
`--coding_only`		Only return consequences that fall in the coding regions of transcripts. *Not used by default*
`--no_intergenic`		Do not include intergenic consequences in the output. *Not used by default*
`--pick`		Pick once line or block of consequence data per variant, including transcript-specific columns. Consequences are chosen according to the criteria described here, and the order the criteria are applied may be customised with --pick_order. This is the best method to use if you are interested only in one consequence per variant. *Not used by default*
`--pick_allele`		Like --pick, but chooses one line or block of consequence data per variant allele. Will only differ in behaviour from --pick when the input variant has multiple alternate alleles. *Not used by default*
`--per_gene`		Output only the most severe consequence per gene. The transcript selected is arbitrary if more than one has the same predicted consequence. Uses the same ranking system as --pick. *Not used by default*
`--pick_allele_gene`		Like --pick_allele, but chooses one line or block of consequence data per variant allele **and** gene combination. *Not used by default*

`--flag_pick`	As per --pick, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*
`--flag_pick_allele`	As per --pick_allele, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*
`--flag_pick_allele_gene`	As per --pick_allele_gene, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*
`--pick_order [c1,c2,...,cN]`	Customise the order of criteria applied when choosing a block of annotation data with e.g. --pick. See this page for the default order. Valid criteria are: *canonical,appris,tsl,biotype,ccds,rank,length*
`--most_severe`	Output only the most severe consequence per variant. Transcript-specific columns will be left blank. Consequence ranks are given in this table. *Not used by default*
`--summary`	Output only a comma-separated list of all observed consequences per variant. Transcript-specific columns will be left blank. *Not used by default*
`--filter_common`	Shortcut flag for the filters below - this will exclude variants that have a co-located existing variant with global AF > 0.01 (1%). May be modified using any of the following freq_* filters. *Not used by default*
`--check_frequency`	Turns on frequency filtering. Use this to include or exclude variants based on the frequency of co-located existing variants in the Ensembl Variation database. You must also specify all of the --freq_* flags below. Frequencies used in filtering are added to the output under the FREQS key in the Extra field. *Not used by default*
`--freq_pop [pop]`	Name of the population to use in frequency filter. This must be one of the following:

Name	Description
1KG_ALL	1000 genomes combined population (global)
1KG_AFR	1000 genomes combined African population
1KG_AMR	1000 genomes combined American population
1KG_EAS	1000 genomes combined East Asian population
1KG_EUR	1000 genomes combined European population
1KG_SAS	1000 genomes combined South Asian population
ESP_AA	NHLBI-ESP African American
ESP_EA	NHLBI-ESP European American
ExAC	ExAC combined population
ExAC_Adj	ExAC combined adjusted population
ExAC_AFR	ExAC African
ExAC_AMR	ExAC American
ExAC_EAS	ExAC East Asian
ExAC_FIN	ExAC Finnish
ExAC_NFE	ExAC non-Finnish European
ExAC_SAS	ExAC South Asian
ExAC_OTH	ExAC other

`--freq_freq [freq]`	Allele frequency to use for filtering. Must be a float value between 0 and 1
`--freq_gt_lt [gt|lt]`	Specify whether the frequency of the co-located variant must be greater than (**gt**) or less than (**lt**) the value specified with --freq_freq
`--freq_filter [exclude|include]`	Specify whether to **exclude** or **include** only variants that pass the frequency filter

Database options

Flag	Alternate	Description
`--database`		Enable VEP to use local or remote databases.

`--host [hostname]`		Manually define the database host to connect to. Users in the US may find connection and transfer speeds quicker using our East coast mirror, useastdb.ensembl.org. *Default = "ensembldb.ensembl.org"*
`--user [username]`	`-u`	Manually define the database username. *Default = "anonymous"*
`--password [password]`	`--pass`	Manually define the database password. *Not used by default*
`--port [number]`		Manually define the database port. *Default = 5306*
`--genomes`		Override the default connection settings with those for the Ensembl Genomes public MySQL server. Required when using any of the Ensembl Genomes species. *Not used by default*
`--lrg`		Map input variants to LRG coordinates (or to chromosome coordinates if given in LRG coordinates), and provide consequences on both LRG and chromosomal transcripts. Not compatible with --offline
`--check_svs`		Checks for the existence of structural variants that overlap your input. Currently requires database access (i.e. not compatible with --offline). *Not used by default*
`--db_version [number]`	`--db`	Force the script to connect to a specific version of the Ensembl databases. Not recommended as there may be conflicts between software and database versions. *Not used by default*
`--registry [filename]`		Defining a registry file overwrites other connection settings and uses those found in the specified registry file to connect. *Not used by default*

Variant Effect Predictor Annotation sources

VEP can use a variety of annotation sources to retrieve the transcript models used to predict consequence types.

- **Cache** - a downloadable file containing all transcript models, regulatory features and variant data for a species
- **GFF or GTF** - use transcript models defined in a tabix-indexed GFF or GTF file
- **Database** - connect to a MySQL database server hosting Ensembl databases

Data from VCF, BED and bigWig files can also be incorporated by VEP's custom annotation feature.

Using a cache is the most efficient way to use VEP; we would encourage you to use a cache wherever possible. Caches are easy to download and set up using the installer. Follow the tutorial for a simple guide.

Caches

Using a cache (--cache) is the fastest and most efficient way to use VEP, as in most cases only a single initial network connection is made and most data is read from local disk. Use offline mode to eliminate all network connections for speed and/or privacy.

Downloading caches

Ensembl creates cache files for every species for each Ensembl release. They can be automatically downloaded and configured using INSTALL.pl.

If interested in RefSeq transcripts you may download an alternate cache file (e.g. homo_sapiens_refseq), or a merged file of RefSeq and Ensembl transcripts (eg homo_sapiens_merged); remember to specify --refseq or --merged when running VEP to use the relevant cache. See documentation for full details.

Manually downloading caches

It is also simple to download and set up caches without using the installer. By default, VEP searches for caches in $HOME/.vep; to use a different directory when running VEP, use --dir_cache.

```
cd $HOME/.vep
curl -O ftp://ftp.ensembl.org/pub/release-88/variation/VEP/homo_sapiens_vep_88_GRCh38.tar.gz
tar xzf homo_sapiens_vep_88_GRCh38.tar.gz
```

FTP directories by species grouping:

- Ensembl
  - Vertebrates
- Ensembl Genomes. **NB:** When using Ensembl Genomes caches, you should use the --cache_version option to specify the relevant Ensembl Genomes version number as these differ from the concurrent Ensembl/VEP version numbers.
  - Bacteria
  - Fungi
  - Metazoa
  - Plants
  - Protists

Limitations of the cache

The cache stores the following information:

- Transcript location, sequence, exons and other attributes
- Gene, protein, HGNC and other identifiers for each transcript (where applicable, limitations apply to RefSeq caches)
- Locations, alleles and frequencies of existing variants
- Regulatory regions
- Predictions and scores for SIFT, PolyPhen

It does not store any information pertaining to, and therefore cannot be used for, the following:

- HGVS names (--hgvs, --hgvsg) - to retrieve these you must additionally point to a FASTA file containing the reference sequence for your species (--fasta)
- Using HGVS notation as input (--format hgvs)
- Using variant identifiers as input (--format id)
- Finding overlapping structural variants (--check_sv)

Enabling one of these options with --cache will cause VEP to warn you in its status output with something like the following:

```
 2011-06-16 16:24:51 - INFO: Database will be accessed when using --hgvs
```

Convert with tabix

For those with Bio::DB::HTS (as set up by INSTALL.pl) or tabix installed on their systems, the speed of retrieving existing co-located variants can be greatly improved by converting the cache files using the supplied script, convert_cache.pl. This replaces the plain-text, chunked variant dumps with a single tabix-indexed file per chromosome. The script is simple to run:

```
perl convert_cache.pl -species [species] -version [vep_version]
```

To convert all species and all versions, use "all":

```
perl convert_cache.pl -species all -version all
```

A full description of the options can be seen using **--help**. When complete, VEP will automatically detect the converted cache and use this in place.

Note that tabix and bgzip must be installed on your system to convert a cache. INSTALL.pl downloads these when setting up Bio::DB::HTS; to enable convert_cache.pl to find them, run:

```
export PATH=${PATH}:${PWD}/htslib
```

Data privacy and offline mode

When using the public database servers, VEP requests transcript and variation data that overlap the loci in your input file. As such, these coordinates are transmitted over the network to a public server, which may not be appropriate for those with sensitive or private data. Users should note that **only** the coordinates are transmitted to the server; no other information is sent.

To run VEP in an offline mode that does not use any network connections, use the flag --offline.

The limitations described above apply absolutely when using offline mode. For example, if you specify --offline and --format id, VEP will report an error and refuse to run:

```
ERROR: Cannot use ID format in offline mode
```

All other features, including the ability to use custom annotations and plugins, are accessible in offline mode.

GFF/GTF files

VEP can use transcript annotations defined in GFF or GTF files. The files must be bgzipped and indexed with tabix, and VEP requires a FASTA file containing the genomic sequence in order to generate transcript models.

Your GFF or GTF file must be sorted in chromosomal order. VEP does not use header lines so it is safe to remove them.

```
grep -v "#" data.gff | sort -k1,1 -k4,4n -k5,5n | bgzip -c > data.gff.gz
tabix -p gff data.gff.gz
./vep -i input.vcf -gff data.gff.gz -fasta genome.fa.gz
```

You may use any number of GFF/GTF files in this way, providing they refer to the same genome. You may also use them in concert with annotations from a cache or database source; annotations are distinguished by the SOURCE field in the VEP output:

```
./vep -i input.vcf -cache -gff data.gff.gz -fasta genome.fa.gz
```

This functionality uses VEP's custom annotation feature, and the --gff flag is a shortcut to:

```
--custom data.gff.gz,,gff
```

You should use the longer form if you wish to customise the name of the GFF as it appears in the SOURCE field and VEP output header.

GFF format expectations

VEP has been tested on GFF files generated by Ensembl and NCBI (RefSeq). Due to inconsistency in the GFF specification and adherence to it, VEP may encounter problems parsing some GFF files. For the same reason, not all transcript biotypes defined in your GFF may be supported by VEP. VEP does not support GFF files with embedded FASTA sequence.

The following entity types (3rd column in the GFF) are supported by VEP. Lines of other types will be ignored; if this leads to an incomplete transcript model, the whole transcript model may be discarded. [Show supported types]

Entities in the GFF are expected to be linked using a key named "parent" or "Parent" in the attributes (9th) column of the GFF. Unlinked entities (i.e. those with no parents **or** children) are discarded. Sibling entities (those that share the same parent) may have overlapping coordinates, e.g. for exon and CDS entities.

Transcripts require a Sequence Ontology biotype to be defined in order to be parsed by VEP. The simplest way to define this is using an attribute named "biotype" on the transcript entity. Other configurations are supported in order for VEP to be able to parse GFF files from NCBI and other sources.

GTF format expectations

The following GTF entity types will be parsed by VEP:

- cds (or CDS)
- stop_codon
- exon
- gene
- transcript

Entities are linked by an attribute named for the parent entity type e.g. exon is linked to transcript by transcript_id, transcript is linked to gene by gene_id.

Transcript biotypes are defined in attributes named "biotype", "transcript_biotype" or "transcript_type". If none of these exist, VEP will attempt to interpret the source field (2nd column) of the GTF as the biotype.

Chromosome synonyms

If the chromosome names used in your GFF/GTF differ from those used in the FASTA or your input VCF, you may see warnings like this when running VEP:

```
WARNING: Chromosome 21 not found in annotation sources or synonyms on line 160
```

To circumvent this you may provide VEP with a synonyms file. A synonym file is included in VEP's cache files, so if you have one of these for your species you can use it as follows:

```
./vep -i input.vcf -cache -gff data.gff.gz -fasta genome.fa.gz -synonyms ~/.vep/homo_sapiens/88_GRCh
```

Limitations

Using a GFF or GTF file as VEP's annotation source limits access to some auxiliary information available when using a cache. Currently most external reference data such as gene symbols, transcript identifiers and protein domains are inaccessible when using only a GFF/GTF file.

VEP's flexibility does allow some annotation types to be replaced. The following table illustrates some examples and alternative means to retrieve equivalent data.

Data type	Alternative
SIFT and PolyPhen predictions (--sift, --polyphen)	Use the PolyPhen_SIFT VEP plugin
Co-located variants (--check_existing, --af* flags)	A couple of options are available: 1. Use a VCF with --custom to retrieve variant IDs, frequency and other data

	2. Add --cache to use variants in the cache. *
Regulatory consequences (--regulatory)	Add --cache to use regulatory features in the cache. *

* Note this will also instruct VEP to annotate input variants against transcript models retrieved from the cache **as well as** those from the GFF/GTF file. It is possible to use --transcript_filter to include only the transcripts from your GFF/GTF file:

```
./vep -i input.vcf -cache -custom data.gff.gz,myGFF,gff -fasta genome.fa.gz -transcript_filter "_sou
```

FASTA files

By pointing VEP to a FASTA file (or directory containing several files), it is possible to retrieve reference sequence locally when using --cache or --offline. This enables VEP to retrieve HGVS notations (--hgvs), check the reference sequence given in input data (--check_ref), and construct transcript models from a GFF or GTF file without accessing a database.

FASTA files can be set up using the installer; files set up using the installer are automatically detected by VEP when using --cache or --offline; you should not need to use --fasta to manually specify them.

To enable this VEP uses one of two modules:

- The Bio::DB::HTS Perl XS module with HTSlib. This module uses compiled C code and can access compressed (bgzipped) or uncompressed FASTA files. It is set up by the VEP installer.
- The Bio::DB::Fasta module. This may be used on systems where installation of the Bio::DB::HTS module has not been possible. It can access only uncompressed FASTA files. It is also set up by the VEP installer and comes as part of the BioPerl package.

The first time you run VEP with a specific FASTA file, an index will be built. This can take a few minutes, depending on the size of the FASTA file and the speed of your system. On subsequent runs the index does not need to be rebuilt (if the FASTA file has been modified, VEP will force a rebuild of the index).

Ensembl provides suitable reference FASTA files as downloads from its FTP server. See the Downloads page for details. You should preferably use the installer as described above to fetch these files; manual instructions are provided for reference. In most cases it is best to download the single large "primary_assembly" file for your species. You should use the unmasked (without "_rm" or "_sm" in the name) sequences. Note that VEP requires that the file be either unzipped (Bio::DB::Fasta) or unzipped and then recompressed with bgzip (Bio::DB::HTS::Faidx) to run; when unzipped these files can be very large (25GB for human). An example set of commands for setting up the data for human follows:

```
curl -O ftp://ftp.ensembl.org/pub/release-88/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_
gzip -d Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
bgzip Homo_sapiens.GRCh38.dna.primary_assembly.fa
./vep -i input.vcf --offline --hgvs --fasta Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
```

Databases

VEP can use remote or local database servers to retrieve annotations.

- Using --cache (without --offline) uses the local cache on disk to fetch most annotations, but allows database connections for some features (see cache limitations)
- Using --database tells VEP to retrieve **all** annotations from the database. **Please only use this for small input files or when using a local database server!**

Public database servers

By default, VEP is configured to connect to Ensembl's public MySQL instance at ensembldb.ensembl.org. For users in the US (or for any user geographically closer to the East coast of the USA than to Ensembl's data centre in Cambridge, UK), a mirror server is available at useastdb.ensembl.org. To use the mirror, use the flag `--host useastdb.ensembl.org`

Users of Ensembl Genomes species (e.g. plants, fungi, microbes) should use their public MySQL instance; the connection parameters for this can be automatically loaded by using the flag --genomes

Users with small data sets (100s of variants) should find using the default connection settings adequate. Those with larger data sets, or those who wish to use VEP in a batch manner, should consider one of the alternatives below.

Using a local database

It is possible to set up a local MySQL mirror with the databases for your species of interest installed. For instructions on installing a local mirror, see here. You will need a MySQL server that you can connect to from the machine where you will run VEP (this can be the same machine). For most of the functionality of VEP, you will only need the Core database (e.g. homo_sapiens_core_88_38) installed. In order to find co-located variants or to use SIFT or PolyPhen, it is also necessary to install the relevant variation database (e.g. homo_sapiens_variation_88_38).

Note that unless you have custom data to insert in the database, in most cases it will be much more efficient to use a pre-built cache in place of a local database.

To connect to your mirror, you can either set the connection parameters using --host, --port, --user and --password, or use a registry file. Registry files contain all the connection parameters for your database, as well as any species aliases you wish to set up:

```
use Bio::EnsEMBL::DBSQL::DBAdaptor;
use Bio::EnsEMBL::Variation::DBSQL::DBAdaptor;
use Bio::EnsEMBL::Registry;

Bio::EnsEMBL::DBSQL::DBAdaptor->new(
  '-species' => "Homo_sapiens",
  '-group'   => "core",
  '-port'    => 5306,
  '-host'    => 'ensembldb.ensembl.org',
  '-user'    => 'anonymous',
  '-pass'    => '',
  '-dbname'  => 'homo_sapiens_core_88_38'
);

Bio::EnsEMBL::Variation::DBSQL::DBAdaptor->new(
  '-species' => "Homo_sapiens",
  '-group'   => "variation",
  '-port'    => 5306,
  '-host'    => 'ensembldb.ensembl.org',
  '-user'    => 'anonymous',
  '-pass'    => '',
  '-dbname'  => 'homo_sapiens_variation_88_38'
);

Bio::EnsEMBL::Registry->add_alias("Homo_sapiens","human");
```

For more information on the registry and registry files, see here.

Cache - technical information

ADVANCED The cache consists of compressed files containing listrefs of serialised objects. These objects are initially created from the database as if using the Ensembl API normally. In order to reduce the size of the cache and allow the serialisation to occur, some changes are made to the objects before they are dumped to disk. This means that they will not behave in exactly the same way as an object retrieved from the database when writing, for example, a plugin that uses the cache.

The following hash keys are deleted from each transcript object:

- **analysis**
- **created_date**
- **dbentries** : this contains the external references retrieved when calling $transcript->get_all_DBEntries(); hence this call on a cached object will return no entries
- **description**
- **display_xref**
- **edits_enabled**
- **external_db**
- **external_display_name**
- **external_name**
- **external_status**
- **is_current**

- **modified_date**
- **status**
- **transcript_mapper** : used to convert between genomic, cdna, cds and protein coordinates. A copy of this is cached separately by VEP as

```
$transcript->{_variation_effect_feature_cache}->{mapper}
```

As mentioned above, a special hash key "_variation_effect_feature_cache" is created on the transcript object and used to cache things used by VEP in predicting consequences, things which might otherwise have to be fetched from the database. Some of these are stored in place of equivalent keys that are deleted as described above. The following keys and data are stored:

- **introns** : listref of intron objects for the transcript. The adaptor, analysis, dbID, next, prev and seqname keys are stripped from each intron object
- **translateable_seq** : as returned by

```
$transcript->translateable_seq
```

- **mapper** : transcript mapper as described above
- **peptide** : the translated sequence as a string, as returned by

```
$transcript->translate->seq
```

- **protein_features** : protein domains for the transcript's translation as returned by

```
$transcript->translation->get_all_ProteinFeatures
```

  Each protein feature is stripped of all keys but: start, end, analysis, hseqname
- **codon_table** : the codon table ID used to translate the transcript, as returned by

```
$transcript->slice->get_all_Attributes('codon_table')->[0]
```

- **protein_function_predictions** : a hashref containing the keys "sift" and "polyphen"; each one contains a protein function prediction matrix as returned by e.g.

```
$protein_function_prediction_matrix_adaptor->fetch_by_analysis_translation_md5('sift', md5_hex($transcrip
```

Similarly, some further data is cached directly on the transcript object under the following keys:

- **_gene** : gene object. This object has all keys but the following deleted: start, end, strand, stable_id
- **_gene_symbol** : the gene symbol
- **_ccds** : the CCDS identifier for the transcript
- **_refseq** : the "NM" RefSeq mRNA identifier for the transcript
- **_protein** : the Ensembl stable identifier of the translation
- **_source_cache** : the source of the transcript object. Only defined in the merged cache (values: Ensembl, RefSeq) or when using a GFF/GTF file (value: short name or filename)

Variant Effect Predictor Filtering results

The filter_vep script is included along side the main VEP script. It can be used to filter VEP output files to find important or interesting results.

It operates on standard, tab-delimited or VCF formatted output (NB only VCF output produced by VEP or in the same format can be used).

Running filter_vep

Run the script as follows:

```
./vep -i in.vcf -o out.txt -cache -everything
./filter_vep -i out.txt -o out_filtered.txt -filter "[filter_text]"
```

The script can also read from STDIN and write to STDOUT, and so may be used in a UNIX pipe:

```
./vep -i in.vcf -o stdout -cache -check_existing | ./filter_vep -filter "not Existing_variation" -o
```

The above command removes known variants from your output

Options

Flag	Alternate	Description
`--help`	`-h`	Print usage message and exit
`--input_file [file]`	`-i`	Specify the input file (i.e. the VEP results file). If no input file is specified, the script will attempt to read from STDIN. Input may be gzipped - to force the script to read a file as gzipped, use --gz
`--format [format]`		Specify input file format (vep or vcf)
`--output_file [file]`	`-o`	Specify the output file to write to. If no output file is specified, the script will write to STDOUT
`--force_overwrite`		Force the script to overwrite the output file if it already exists
`--filter [filters]`	`-f`	Add filter (see below). Multiple --filter flags may be used, and are treated as logical ANDs, i.e. all filters must pass for a line to be printed
`--list`	`-l`	List allowed fields from the input file
`--count`	`-c`	Print only a count of matched lines
`--only_matched`		In VCF files, the CSQ field that contains the consequence data will often contain more than one "block" of consequence data, where each block corresponds to a variant/feature overlap. Using --only_matched will remove blocks that do not pass the filters. By default, the script prints out the entire VCF line if any of the blocks pass the filters.
`--ontology`	`-y`	Use Sequence Ontology to match consequence terms. Use with operator "is" to match against all child terms of your value. e.g. "Consequence is coding_sequence_variant" will match missense_variant, synonymous_variant etc. Requires database connection; defaults to connecting to ensembldb.ensembl.org. Use --host, --port, --user, --password, --version as per **vep** to change connection parameters.

Writing filters

Filter strings consist of three components:

1. **Field** : A field name from the VEP results file. This can be any field in the "main" columns of the output, or any in the "Extra" final column. For VCF files, this is any field defined in the "##INFO=<ID=CSQ" header. You can list available fields using --list. Field names are not case sensitive, and you may use the first few characters of a field name if they resolve uniquely to one field name.
2. **Operator** : The operator defines the comparison carried out.
3. **Value** : The value to which the content of the field is compared. May be prefixed with "#" to represent the value of another field.

Examples:

```
# match entries where Feature (Transcript) is "ENST00000307301"
--filter "Feature is ENST00000307301"

# match entries where Protein_position is less than 10
--filter "Protein_position < 10"

# match entries where Consequence contains "stream" (this will match upstream and downstream)
--filter "Consequence matches stream"
```

For certain fields you may only be interested in whether a value exists for that field; in this case the operator and value can be left out:

```
# match entries where the gene symbol is defined
--filter "SYMBOL"
```

The value component may be another field; to represent this, prefix the name of the field to be used as a value with "#":

```
# match entries where AFR_AF is greater than EUR_AF
--filter "AFR_AF > #EUR_AF"
```

Filter strings can be linked together by the logical operators "or" and "and", and inverted by prefixing with "not":

```
# filter for missense variants in CCDS transcripts where the variant falls in a protein domain
--filter "Consequence is missense_variant and CCDS and DOMAINS"

# find variants where the allele frequency is greater than 10% in either AFR or EUR populations
--filter "AFR_AF > 0.1 or EUR_AF > 0.1"

# filter out known variants
--filter "not Existing_variation"
```

Filter logic may be constrained using parentheses, to any arbitrary level:

```
# find variants with AF > 0.1 in AFR or EUR but not EAS or SAS
--filter "(AFR_AF > 0.1 or EUR_AF > 0.1) and (EAS_AF < 0.1 and SAS_AF < 0.1)"
```

For fields that contain string and number components, the script will try and match the relevant part based on the operator in use. For example, using --sift b in VEP gives strings that look like "tolerated(0.46)". This will give a match to either of the following filters:

```
# match string part
--filter "SIFT is tolerated"

# match number part
--filter "SIFT < 0.5"
```

Note that for numeric fields, such as the *AF allele frequency fields, filter_vep does not consider the absence of a value for that field as equivalent to a 0 value. For example, if you wish to find rare variants by finding those where the allele frequency is less than 1% **or** absent, you should use the following:

```
--filter "AF < 0.01 or not AF"
```

For the Consequence field it is possible to use the Sequence Ontology to match terms ontologically; for example, to match all coding consequences (e.g. missense_variant, synonymous_variant):

```
--ontology --filter "Consequence is coding_sequence_variant"
```

Operators

- **is** (synonyms: = , eq) : Match exactly
  ```
  # get only transcript consequences
  --filter "Feature_type is Transcript"
  ```
- **!=** (synonym: ne) : Does not match exactly
  ```
  # filter out tolerated SIFT predictions
  --filter "SIFT != tolerated"
  ```
- **match** (synonyms: matches , re , regex) : Match string using regular expression. You may include any regular expression notation, e.g. "\d" for any numerical character
  ```
  # match stop_gained, stop_lost and stop_retained
  --filter "Consequence match stop"
  ```
- **<** (synonym: lt) : Less than. Note an absent value is not considered to be equivalent to 0.
  ```
  # find SIFT scores less than 0.1
  --filter "SIFT < 0.1"
  ```
- **>** (synonym: gt) : Greater than
  ```
  # find variants not in the first exon
  --filter "Exon > 1"
  ```
- **<=** (synonym: lte) : Less than or equal to. Note an absent value is not considered to be equivalent to 0.
- **>=** (synonym: gte) : Greater than or equal to
- **exists** (synonyms: ex , defined) : Field is defined - equivalent to using no operator and value
- **in** : Find in list or file. Value may be either a comma-separated list or a file containing values on separate lines. Each list item is compared using the "is" operator.
  ```
  # find variants in a list of gene names
  --filter "SYMBOL in BRCA1,BRCA2"

  # filter using a file of MotifFeatures
  --filter "Feature in /data/files/motifs_list.txt"
  ```

Variant Effect Predictor Custom annotations

VEP can integrate custom annotation from standard format files into your results by using the --custom flag.

These files may be hosted locally or remotely, with no limit to the number or size of the files. The files must be indexed using the tabix utility (BED, GFF, GTF, VCF); bigWig files contain their own indices.

Annotations typically appear as key=value pairs in the Extra column of the VEP output; they will also appear in the INFO column if using VCF format output. The value for a particular annotation is defined as the identifier for each feature; if not available, an identifier derived from the coordinates of the annotation is used. Annotations will appear in each line of output for the variant where multiple lines exist.

Data formats

VEP supports the following formats:

- Gene/transcript annotations. Requires FASTA file; see documentation.
    - GFF : a format for describing genes and other genomic features.
    - GTF : a similar format derived from GFF.
- Variant data
    - VCF : a format used to describe genomic variants. VEP will use the 3rd column of the file as the identifier. INFO fields from records may be added to the VEP output.
- Basic/uninterpreted data
    - BED : a simple tab-delimited format containing 3-12 columns of data. The first 3 columns contain the coordinates of the feature. If available, VEP will use the 4th column of the file as the identifier of the feature.
    - bigWig : a format for storage of dense continuous data. VEP uses the value for the given position as the "identifier". Note that bigWig files contain their own indices, and do not need to be indexed by tabix. Requires Bio::DB::BigFile.

Any other files can be easily converted to be compatible with VEP; the easiest format to produce is a BED-like file containing coordinates and an (optional) identifier:

```
chr1    10000    11000    Feature1
chr3    25000    26000    Feature2
chrX    99000    99001    Feature3
```

Chromosomes can be denoted by either e.g. "chr7" or "7", "chrX" or "X".

Preparing files

Custom annotation files must be prepared in a particular way in order to work with tabix and therefore with VEP. Files must be sorted in chromosome and position order, compressed using bgzip and finally indexed using tabix. Here is an example of that process for a BED file:

```
sort -k1,1 -k2,2n -k3,3n myData.bed | bgzip > myData.bed.gz
tabix -p bed myData.bed.gz
```

The tabix utility has several preset filetypes that it can process, and it can also process any arbitrary filetype containing at least a chromosome and position column. See the documentation for details.

If you are going to use the file remotely (i.e. over HTTP or FTP protocol), you should ensure the file is world-readable on your server.

Options

Each custom file that you configure VEP to use can be configured. Beyond the filepath, there are further options, each of which is specified in a comma-separated list, for example:

```
./vep -custom frequencies.bw,Frequency,bigwig,exact,0
./vep -custom http://www.myserver.com/data/myPhenotypes.bed.gz,Phenotype,bed,exact,1
```

The options are as follows:

- **Filename** : The path to the file. For tabix indexed files, the VEP will check that both the file and the corresponding .tbi file exist. For remote files, VEP will check that the tabix index is accessible on startup.
- **Short name** : A name for the annotation that will appear as the key in the key=value pairs in the results. If not defined, this will default to e.g. "Custom1" for the first set of annotation added.
- **File type** : One of "bed", "gff", "gtf", "vcf", "bigwig". If not specified, VEP assumes the file is BED format.
- **Annotation type** : One of "exact", "overlap". When using "exact" only annotations whose coordinates match exactly those of the variant will be reported. This would be suitable for position specific information such as conservation scores, allele frequencies or phenotype information. Using "overlap", any annotation that overlaps the variant by even 1bp will be reported. Only "overlap" is supported for GFF and GTF formats.
- **Force report coordinates** : One of "0" or "1" (if left blank assumed to be "0") - if set to "1", this forces VEP to output the coordinates of an overlapping custom feature instead of any found identifier (or value in the case of bigWig) field. If set to "0" (the default), VEP will output the identifier field if one is found; if none is found, then the coordinates are used instead.
- **VCF fields** : if any field names are specified that are found in the info field of the VCF, these will also be added as custom annotations. If using "exact" annotation type, allele-specific annotation will be retrieved. The INFO field name will be prefixed with the short name, e.g. using short name "test", the INFO field "foo" will appear as "test_FOO" in the VEP output.

All options (apart from the filename) are optional and their absence will invoke the default behaviour.

Using remote files

The tabix utility makes it possible to read annotation files from remote locations, for example over HTTP or FTP protocols. In order to do this, the .tbi index file is downloaded locally (to the current working directory) when VEP is run. From this point on, only the portions of data requested by VEP (i.e. those overlapping the variants in your input file) are downloaded. Users should be aware, however, that it is still possible to cause problems with network traffic in this manner by requesting data for a large number of variants. Users with large amounts of data should download the annotation file locally rather than risk causing any issues!

bigWig files can also be used remotely in the same way as tabix-indexed files, although less stringent checks are carried out on VEP startup.

Variant Effect Predictor Plugins

VEP can use plugin modules written in Perl to add functionality to the software. Plugins are a powerful way to extend, filter and manipulate the output of VEP.

Example functionality:

- Running additional Ensembl API code (Conservation, LD)
- Querying an external dataset or database (CADD, dbNSFP, ExAC)
- Running external algorithms (Condel, MaxEntScan)
- Filtering VEP output (RankFilter)

Plugins can be installed using VEP's installer script, run the following command to get a list of available plugins:

```
perl INSTALL.pl -a p -g list
```

Some plugins are also available to use via the VEP web interface.

Examples

We have written several example plugins that implement experimental functionality that we do not (yet) include in the variation API, and these are stored in a public github repository:

https://github.com/Ensembl/VEP_plugins

We hope that these will serve as useful examples for users implementing new plugins. If you have any questions about the system, or suggestions for enhancements please let us know on the ensembl-dev mailing list. We also encourage users to share any plugins they develop; we are happy to accept pull requests on the VEP_plugins git repository.

If you have VEP plugins or other code to share with the community, Ensembl e!code is a directory for extensions to Ensembl.

How it works

Plugins are run once VEP has finished its analysis for each line of the output, but before anything is printed to the output file. When each plugin is called (using the 'run' method) it is passed two data structures to use in its analysis; the first is a data structure containing all the data for the current line, and the second is a reference to a variation API object that represents the combination of a variant allele and an overlapping or nearby genomic feature (such as a transcript or regulatory region). This object provides access to all the relevant API objects that may be useful for further analysis by the plugin (such as the current VariationFeature and Transcript); please refer to the variation API documentation for more details.

Functionality

We expect that most plugins will simply add information to the last column of the output file, the "Extra" column, and the plugin system assumes this in various places, but plugins are also free to alter the output line as desired.

The only hard requirement for a plugin to work with VEP is that it implements a number of required methods (such as 'new' which should create and return an instance of this plugin, 'get_header_info' which should return descriptions of the type of data this plugin produces to be included in VEP output's header, and 'run' which should actually perform the logic of the plugin). To make development of plugins easier, we suggest that users use the Bio::EnsEMBL::Variation::Utils::BaseVepPlugin module as their base class, which provides default implementations of all the necessary methods which can be overridden as required. Please refer to the documentation in this module for details of all required methods and for a simple example of a plugin implementation.

Filtering using plugins

A common use for plugins will be to filter the output in some way (for example to limit output lines to missense variants) and so we provide a simple mechanism to support this. The 'run' method of a plugin is assumed to return a reference to a hash containing information to be included in the output, and if a plugin should not add any data to a particular line it should return an empty hashref. If a plugin should instead filter a line and exclude it from the output, it should return 'undef' from its 'run' method, this also means that no further plugins will be run on the line. If you are developing a filter plugin, we suggest that you use the Bio::EnsEMBL::Variation::Utils::BaseVepFilterPlugin as your base class and then you need only override the 'include_line' method to return true if you want to include this line, and false otherwise. Again, please refer to the documentation in this module for more details and an example implementation of a missense filter.

Using plugins

In order to run a plugin you need to include the plugin module in Perl's library path somehow; by default VEP includes the '~/.vep/Plugins' directory in the path, so this is a convenient place to store plugins, but you are also able to include modules by any other means (e.g using the $PERL5LIB environment variable in Unix-like systems). You can then run a plugin using the --plugin command line option, passing the name of the plugin module as the argument. For example, if your plugin is in a module called MyPlugin.pm, stored in ~/.vep/Plugins, you can run it with a command line like:

```
./vep -i input.vcf --plugin MyPlugin
```

You can pass arguments to the plugin's 'new' method by including them after the plugin name on the command line, separated by commas, e.g.:

```
./vep -i input.vcf --plugin MyPlugin,1,FOO
```

If your plugin inherits from BaseVepPlugin, you can then retrieve these parameters as a list from the 'params' method.

You can run multiple plugins by supplying multiple --plugin arguments. Plugins are run serially in the order in which they are specified on the command line, so they can be run as a pipeline, with, for example, a later plugin filtering output based on the results from an earlier plugin. Note though that the first plugin to filter a line 'wins', and any later plugins won't get run on a filtered line.

Intergenic variants

When a variant falls in an intergenic region, it will usually not have any consequence types called, and hence will not have any associated VariationFeatureOverlap objects. In this special case, VEP creates a new VariationFeatureOverlap that overlaps a feature of type "Intergenic". To force your plugin to handle these, you must add "Intergenic" to the feature types that it will recognize; you do this by writing your own feature_types sub-routine:

```
sub feature_types {
    return ['Transcript', 'Intergenic'];
}
```

This will cause your plugin to handle any variation features that overlap transcripts or intergenic regions. To also include any regulatory features, you should use the generic type "Feature":

```
sub feature_types {
    return ['Feature', 'Intergenic'];
}
```

Example commands

- Read input from STDIN, output to STDOUT

```
./vep -cache -o stdout
```

- Add regulatory region consequences

```
./vep -cache -i variants.txt -regulatory
```

- Input file variants.vcf.txt, input file format VCF, add gene symbol identifiers

```
./vep -cache -i variants.vcf.txt -format vcf -symbol
```

- Filter out common variants based on 1000 genomes data

```
./vep -cache -i variants.txt -filter_common
```

- Force overwrite of output file variants_output.txt, check for existing co-located variants, output only coding sequence consequences, output HGVS names

```
./vep -cache -i variants.txt -o variants_output.txt -force -check_existing -coding_only -hgvs
```

- Specify DB connection parameters in registry file ensembl.registry, add SIFT score and prediction, PolyPhen prediction

```
./vep -database -i variants.txt -registry ensembl.registry -sift b -polyphen p
```

- Connect to Ensembl Genomes db server for A.thaliana

```
./vep -database -i variants.txt -genomes -species arabidopsis_thaliana
```

- Load config from ini file, run in quiet mode

```
./vep -config vep.ini -i variants.txt -q
```

- Use cache in /home/vep/mycache/, use gzcat instead of zcat

```
./vep -cache -dir /home/vep/mycache/ -i variants.txt -compress gzcat
```

- Add custom position-based phenotype annotation from remote BED file

```
./vep -cache -i variants.vcf -custom ftp://ftp.myhost.org/data/phenotypes.bed.gz,phenotype
```

- Use the plugin named MyPlugin, output only the variation name, feature, consequence type and MyPluginOutput fields

```
./vep -cache -i variants.vcf -plugin MyPlugin -fields Uploaded_variation,Feature,Consequence,MyP
```

gnomAD

ExAC data is available embedded in the cache files for human. ExAC's successor project, gnomAD contains double the number of exomes plus a large number of whole genomes. This data will be added in a future release of Ensembl/VEP, but it is simple to have VEP access this data now:

1. Ensure your system is set up to access data from tabix-indexed VCF files (see documentation)
2. Download the (exomes) file and tabix index:
   - VCF
   - tabix index
3. Run VEP with the following command (using the GRCh37 input example) to get locations and continental-level allele frequencies:

```
./vep -i examples/homo_sapiens_GRCh37.vcf -cache \
-custom gnomad.exomes.r2.0.1.sites.vcf.gz,gnomAD,vcf,exact,0,AF_AFR,AF_AMR,AF_ASJ,AF_EAS,AF_FIN,
```

You will then see data under field names as described in the VEP output header:

```
## gnomAD : gnomad.exomes.r2.0.1.sites.vcf.gz (exact)
## gnomAD_AFR_AF : AFR_AF field from gnomad.exomes.r2.0.1.sites.vcf.gz
## gnomAD_AMR_AF : AMR_AF field from gnomad.exomes.r2.0.1.sites.vcf.gz
...
```

where the gnomAD field contains the ID (or coordinates if no ID found) of the variant in the VCF file. Any of the fields in the gnomAD file INFO field can be added by appending them to the list in your VEP command.

Note that the above data are mapped to GRCh37. The gnomAD VCF can be mapped to GRCh38 using CrossMap.

Conservation scores

You can use VEP's custom annotation feature to add conservation scores to your output. For example, to add GERP scores, download the bigWig file from the list below, and run VEP with the following flag:

```
./vep -cache -i example.vcf -custom All_hg19_RS.bw,GERP,bigwig
```

Example conservation score files:

- Human (GRCh38)
  - phastCons 7-way
  - phastCons 20-way
  - phastCons 100-way
  - phyloP 7-way
  - phyloP 20-way
  - phyloP 100-way
- Human (GRCh37)
  - GERP
  - phastCons 46-way
  - phastCons 100-way
  - phyloP 46-way
  - phyloP 100-way

All files provided by the UCSC genome browser - files for other species are available from their FTP site, though be sure to use the file corresponding to the correct assembly.

dbNSFP

dbNSFP - "a lightweight database of human nonsynonymous SNPs and their functional predictions" - provides pathogenicity predictions from many tools (including SIFT, PolyPhen, LRT, MutationTaster, FATHMM) across every possible missense substitution in the human proteome. The data is available to download, and while it cannot be immediately used by the VEP it is simple to process the data into a format that the dbNSFP.pm plugin can use.

After downloading the file, you will need to process it so that tabix can index it correctly. This will take a while as the file is very large! Note that you will need the tabix utility in your path to use dbNSFP.

```
unzip dbNSFP3.3a.zip
head -n1 dbNSFP3.3a_variant.chr1 > dbNSFP3.3a.txt
cat dbNSFP3.3a_variant.chr* | grep -v "#" >> dbNSFP3.3a.txt
rm dbNSFP2.0_variant.chr*
bgzip dbNSFP3.3a.txt
tabix -s 1 -b 2 -e 2 dbNSFP3.3a.txt.gz
```

Then simply download the dbNSFP VEP plugin and place it either in **$HOME/.vep/Plugins/** or a path in your **$PERL5LIB**. When you run VEP with the plugin, you will need to select some of the columns that you wish to retrieve; to list them run VEP with the plugin and the path to the dbNSFP file and no further parameters:

```
./vep -cache -force -plugin dbNSFP,dbNSFP3.3a.txt.gz
2014-04-04 11:27:05 - Read existing cache info
2014-04-04 11:27:05 - Auto-detected FASTA file in cache directory
2014-04-04 11:27:05 - Checking/creating FASTA index
2014-04-04 11:27:05 - Failed to instantiate plugin dbNSFP: ERROR: No columns selected to fetch. Avai
#chr,pos(1-coor),ref,alt,aaref,aaalt,hg18_pos(1-coor),genename,Uniprot_acc,
Uniprot_id,Uniprot_aapos,Interpro_domain,cds_strand,refcodon,SLR_test_statistic,
codonpos,fold-degenerate,Ancestral_allele,Ensembl_geneid,Ensembl_transcriptid,
...
```

Note that some of these fields are replicates of those produced by the core VEP code (e.g. SIFT, PolyPhen, the 1000 Genomes and ESP frequencies) - you should use the options to enable these from the VEP code in place of the annotations from dbNSFP as the dbNSFP file covers **only** missense substitutions. Other fields, such as the conservation scores, may be better served by using genome-wide files as described above.

To select fields, just add them as a comma-separated list to your command line:

```
./vep -cache -force -plugin dbNSFP,dbNSFP3.3a.txt.gz,LRT_score,FATHM_score,MutationTaster_score
```

One final point to note is that the dbNSFP scores are frozen on a particular Ensembl release's transcript set; check the readme file on their download site to find out exactly which. While in the majority of cases protein sequences don't change between releases, in some circumstances the protein sequence used by VEP in the latest release may differ from the sequence used to calculate the scores in dbNSFP.

Citations and VEP users

VEP is used by many organisations and projects:

- VEP forms a part of Illumina's VariantStudio software
- Gemini is a framework for exploring genome variation that uses VEP
- The DECIPHER project uses VEP in its analysis pipelines

Other citations and use cases:

- VAX is a suite of plugins for VEP that expands its functionality
- pViz is a visualisation tool for VEP results files
- McCarthy *et al* compares VEP to AnnoVar
- Pabinger *et al* reviews variant analysis software, including VEP
- VEP is used to provide annotation for the ExAC and gnomAD projects

Getting VEP to run faster

Set up correctly, VEP is capable of processing around 3 million variants in 30 minutes. There are a number of steps you can take to make sure your VEP installation is running as fast as possible:

1. Make sure you have the latest version of VEP and the Ensembl API. We regularly introduce optimisations, alongside the new features and bug fixes of a typical new release.

2. Download a cache file for your species. If you are using --database, you should consider using --cache or --offline instead. Any time VEP has to access data from the database (even if you have a local copy), it will be slower than accessing data in the cache on your local file system.

   Enabling certain flags forces VEP to access the database, and the script will warn you at startup that it will do this with e.g.:

   ```
   2011-06-16 16:24:51 - INFO: Database will be accessed when using --check_svs
   ```

   Consider carefully whether you need to use these flags in your analysis.

3. If you use --check_existing or any flags that invoke it (e.g. --af, --af_1kg, --filter_common, --everything), tabix-convert your cache file. Checking for known variants using a converted cache is >100% faster than using the default format.

4. Download a FASTA file if you use --hgvs or --check_ref. Again, this will prevent VEP accessing the database unnecessarily (in this case to retrieve genomic sequence).

5. Using forking enables VEP to run multiple parallel "threads", with each thread processing a subset of your input. Most modern computers have more than one processor core, so running VEP with forking enabled can give huge speed increases (3-4x faster in most cases). Even computers with a single core will see speed benefits due to overheads associated with using object-oriented code in Perl.

   To use forking, you must choose a number of forks to use with the --fork flag. Most users should use 4 forks:

   ```
   ./vep -i my_input.vcf -fork 4 -offline
   ```

   but depending on various factors specific to your setup you may see faster performance with fewer or more forks.

   VEP users writing plugins should be aware that while the VEP code attempts to preserve the state of any plugin-specific cached data between separate forks, there may be situations where data is lost. If you find this is the case, you should disable forking in the new() method of your plugin by deleting the "fork" key from the $config hash.

6. Make sure your cache and FASTA files are stored on the fastest file system or disk you have available. If you have a lot of memory in your machine, you can even pre-copy the files to memory using tmpfs.

7. Consider if you need to generate HGVS notations (--hgvs); this is a complex annotation step that can add ~50-80% to your runtime. Note also that --hgvs is switched on by --everything.

8. Install the Set::IntervalTree perl package. This package speeds up VEP's internals by changing how overlaps between variants and transcript components are calculated.

9. Install the Ensembl::XS package. This contains compiled versions of certain key subroutines used in VEP that will run faster than the default native Perl equivalents. Using this should improve runtime by 5-10%.

10. Add the --no_stats flag. Calculating statistics adds some runtime to VEP and most users will not need them.

11. VEP is optimised to run on input files that are sorted in chromosomal order. Unsorted files will still work, albeit more slowly.

12. For very large files (for example those from whole-genome sequencing), VEP process can be easily parallelised by dividing your file into chunks (e.g. by chromosome). VEP will also work with tabix-indexed, bgzipped VCF files, and so the tabix utility could be used to divide the input file:

    ```
    tabix -h variants.vcf.gz 12:1000000-20000000 | ./vep -cache -vcf
    ```

Species with multiple assemblies

With the arrival of GRCh38, Ensembl now supports two different assembly versions for the human genome while users transition from GRCh37. We provide a VEP cache download on the latest software version (88) for both assembly versions.

The VEP installer will install and set up the correct cache and FASTA file for your assembly of interest. If using the --AUTO functionality to install without prompts, remember to add the assembly version required using e.g. "--ASSEMBLY GRCh37". It is also possible to have concurrent installations of caches from both assemblies; just use the --assembly to select the correct one when you run VEP.

Once you have installed the relevant cache and FASTA file, you are then able to use VEP as normal. For those using GRCh37 and requiring database access in addition to the cache (for example, to look up variant identifiers using --format id, see cache limitations), the script will warn you that you must change the database port in order to connect to the correct database:

```
ERROR: Cache assembly version (GRCh37) and database or selected assembly version (GRCh38) do not mat

If using human GRCh37 add "--port 3337" to use the GRCh37 database, or --offline to avoid database c
```

For users looking to move their data between assemblies, Ensembl provides an assembly converter tool - if you've downloaded VEP, then you have it already! The script is found in the ensembl-tools/scripts/assembly_converter folder. There is also an online version of the tool available. Both UCSC (liftOver) and NCBI (Remap) also provide tools for converting data between assemblies.

Summarising annotation

By default VEP is configured to provide annotation on every genomic feature that each input variant overlaps. This means that if a variant overlaps a gene with multiple alternate splicing variants (transcripts), then a block of annotation for each of these transcripts is reported in the output. In the default VEP output format each of these blocks is written on a single line of output; in VCF output format the blocks are separated by commas in the INFO field.

For many users, however, this depth of annotation is not required, and to this end VEP provides a number of options to reduce the amount of output produced. Which to choose depends on your motivations and requirements on the output.

NB: Wherever possible we would discourage users from summarising data in this way. Summarising inevitably involves data loss, and invariably at some point this will lead to the loss of biologically relevant information. For example, if your variant overlaps both a regulatory feature and a transcript and you use one of the flags below, the overlap with the regulatory feature will be lost in your output, when in some cases this may be a clue to the "real" functional effect of your variant. For these reasons we encourage users to use one of the flagging options (--flag_pick, --flag_pick_allele or --flag_pick_allele_gene) and to post-filter results.

- **--pick**: this is the option we anticipate will be of use to most users. VEP chooses one block of annotation per variant, using an ordered set of criteria. This order may be customised using --pick_order.

  1. canonical status of transcript
  2. APPRIS isoform annotation
  3. transcript support level
  4. biotype of transcript (protein_coding preferred)
  5. CCDS status of transcript
  6. consequence rank according to this table
  7. translated, transcript or feature length (longer preferred)

  Note that some categories may not be available for the species or cache version that you are using; in these cases the category will be skipped and the next in line used.

- **--pick_allele**: as above, but chooses one consequence block per variant allele. This can be useful for VCF input files with more than one ALT allele
- **--per_gene**: as --pick, but chooses one annotation block per gene that the input variant overlaps
- **--pick_allele_gene**: as above, but chooses one consequence block per variant allele and gene combination.
- **--flag_pick**: instead of choosing one block and removing the others, this option adds a flag "PICK=1" to picked annotation block, allowing users to easily filter on this later using VEP's filtering script
- **--flag_pick_allele**: as above, but flags one block per allele
- **--flag_pick_allele_gene**: as above, but flags one block per allele and gene combination
- **--most_severe**: this flag reports only the consequence type of the block with the highest rank, according to this table. Feature-specific annotation is absent from the output using this flag, so use with caution!
- **--summary**: this flag reports only a comma-separated list of the consequence types predicted for this variant. Feature-specific annotation is absent from the output using this flag, so use with caution!

HGVS notations

Output

HGVS notations can be produced by VEP using the --hgvs flag. Coding (c.) and protein (p.) notations given against Ensembl identifiers use versioned identifiers that guarantee the identifier refers always to the same sequence.

Genomic HGVS notations may be reported using --hgvsg. Note that the named reference for HGVSg notations will be the chromosome name from the user input (as opposed to the officially recommended chromosome accession).

HGVS notations for insertions or deletions are by default shifted 3-prime relative to the reported transcript or protein sequence in accordance with HGVS specifications. This may lead to discrepancies between the coordinates reported in the HGVS nomenclature and the coordinate columns reported by VEP. You may instruct VEP not to shift using --shift_hgvs 0.

Input

VEP supports using HGVS notations as input. This feature is currently under development, and not all HGVS notation types are supported. Notations relative to genomic (g.) or coding (c.) sequences are currently fully supported; protein (p.) notations are supported in limited fashion due to the complexity involved in determining the multiple possible underlying genomic sequence changes that could produce a single protein change. The script will warn the user if it fails to parse a particular notation.

By default VEP uses Ensembl transcripts as its reference for determining consequences, and hence also for HGVS notations. However, it is possible to parse HGVS notations that use RefSeq transcripts as the reference sequence by using the --refseq flag when running the script. Such notations must include the version number of the transcript e.g.

```
NM_080794.3:c.1001C>T
```

where ".3" denotes that this is version 3 of the transcript NM_080794. See below for more details on how VEP can use RefSeq transcripts.

RefSeq transcripts

Ensembl produces databases containing alignments of RefSeq transcript objects to the reference genome, named otherfeatures databases. The otherfeatures databases are used to build the RefSeq cache, and merged with the standard Ensembl core database to produce the merged cache. These caches also contain alignments of CCDS transcripts and Ensembl EST sequences - they may be included in your analysis using --all_refseq.

By using the --refseq flag when running VEP, these alternative transcripts will be used as the reference for predicting variant consequences. Gene IDs given in the output when using this option are generally NCBI GeneIDs.

Users may wish to exclude predicted RefSeq transcripts (those with identifiers beginning with "XM_" or "XR_") by using --exclude_predicted.

Identifiers and other data

VEP's RefSeq cache lacks many classes of data present in the Ensembl transcript cache.

- Included in the RefSeq cache
  - Gene symbol
  - SIFT and PolyPhen predictions
- **Not** included in the RefSeq cache
  - APPRIS annotation
  - TSL annotation
  - Uniprot identifiers
  - CCDS identifiers
  - Protein domains
  - Gene-phenotype association data

Differences to the reference genome

Users should note that RefSeq sequences may differ from the reference genome sequence to which they are aligned. Ensembl's API (and hence VEP) constructs transcript models using the genomic reference sequence. Differences between the RefSeq sequence and the genomic sequence are not accounted for, so the genomic sequence will be used, meaning that some annotations produced by VEP on these transcripts may be inaccurate. Most differences occur in non-coding regions, typically in UTRs at either end of transcripts or in the addition of a poly-A tail, meaning minimal impact on VEP's annotations.

For the GRCh38 VEP cache, each RefSeq transcript is annotated with the REFSEQ_MATCH flag indicating whether and how the RefSeq model differs from the underlying genome. Note that currently the REFSEQ_MATCH flag will not be set when using the GRCh37 cache.

Correcting transcript models with BAM files

WARNING: Experimental feature! NCBI have released BAM files that contain alignments of RefSeq transcripts to the genome. VEP can use these files to correct the models as read from the RefSeq or merged cache, as well as from RefSeq GFF files.

```
./vep -cache -refseq -i variants.vcf -bam interim_GRCh38.p10_knownrefseq_alignments_2017-01-13.bam
```

BAM files are available from NCBI:

- Human GRCh38.p10
- Human GRCh37.p13

NB: The BAM index files (.bai) in this directory are required and will need to be renamed as the perl library used to parse the files expects the index to be named [indexed_bam_file].bai:

```
mv interim_GRCh38.p10_knownrefseq_alignments_2017-01-13.bai interim_GRCh38.p10_knownrefseq_alignment
```

VEP uses the sequence and alignment in the BAM to correct the RefSeq model. If the corrected model does not match the original RefSeq sequence in the BAM, the corrected model is discarded. The success or failure of the BAM edit is recorded in the BAM_EDIT field of the VEP output.

Using a BAM causes VEP to change how alleles are interpreted from input variants. Input variants are typically encoded in VCFs that are called using the reference genome. This means that the alternate (ALT) allele as given in the VCF may correspond to the reference allele as found in the corrected RefSeq transcript model. VEP will account for this, using the corrected reference allele when calculating consequences, and the GIVEN_REF and USED_REF fields in the VEP output indicate any change made. Note that this may clash with any interpretation from using --check_ref, so it is recommended to avoid using this flag.

Variant Effect Predictor FAQ

For any questions not covered here, please send an email to the Ensembl developer's mailing list (public) or contact the Ensembl Helpdesk (private).

General questions

Q: Why don't I see any co-located variations when using species X?

A: Ensembl only has variation databases for a subset of all Ensembl species - see this document for details.

Q: Why has my insertion/deletion variant encoded in VCF disappeared from the VEP output?

A: Ensembl treats unbalanced variants differently to VCF - your variant hasn't disappeared, it may have just changed slightly! You can solve this by giving your variants a unique identifier in the third column of the VCF file. See here for a full discussion.

Q: Why do I see so many lines of output for each variant in my input?

A: While it can be convenient to search for a easy, one word answer to the question "What is the consequence of this variant?", in reality biology does not make it this simple! Many genes have more than one transcript, so VEP provides a prediction for each transcript that a variant overlaps. The VEP script can help here; the --canonical and --ccds options indicate which transcripts are canonical and belong to the CCDS set respectively, while --pick, --per_gene, --summary and --most_severe allow you to give a more summary level assessment per variant.

Furthermore, several "compound" consequences are also possible - if, for example, a variant falls in the final few bases of an exon, it may be considered to affect a splicing site, in addition to possibly affecting the coding sequence.

Since we cannot possibly predict the exact biology of what will happen, what we provide is the most conservative estimate that covers all reasonable scenarios. It is up to you, the user, to interpret this information!

Web VEP questions

Q: How do I access the web version of the Variant Effect Predictor?

A: You can find the web VEP on the Tools page.

Q: Why is the output I get for my input file different when I use the web VEP and the VEP script?

A: Ensure that you are passing equivalent arguments to the script that you are using in the web version. If you are sure this is still a problem, please report it on the ensembl-dev mailing list.

VEP script questions

Q: How can I make VEP run faster?

There are a number of factors that influence how fast VEP runs. Have a look at our handy guide for tips on improving VEP runtime.

Q: Why do I see "N" as the reference allele in my HGVS strings?

Q: Why do I see the following error (or similar) in my VEP output?

```
substr outside of string at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/EnsEMBL/Variatic
Use of uninitialized value $ref_allele in string eq at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/m
Use of uninitialized value in concatenation (.) or string at /nfs/users/nfs_w/wm2/Perl/ensembl-varia
```

Both of these error types are usually seen when using a FASTA file for retrieving sequence. There are a couple of steps you can take to try to remedy them:

1. The index alongside the FASTA can become corrupted. Delete [fastafile].index and re-run VEP to regenerate it. By default this file is located in your $HOME/.vep/[species]/[version]_[assembly] directory.
2. The FASTA file itself may have been corrupted during download; delete the fasta file and the index and re-download (you can use the VEP installer to do this).
3. Older versions of BioPerl (1.2.3 in particular is known to have this) cannot properly index large FASTA files. Make sure you are using a later (>=1.6) version of BioPerl. The VEP installer installs 1.6.1 for you.

If you still see problems after taking these steps, or if you were not using a FASTA file in the first place, please contact us.

Q: Why do I see the following warning?

```
WARNING: Chromosome 21 not found in annotation sources or synonyms on line 160
```

This can occur if the chromsome names differ between your input variant and any annotation source that you are using (cache, database, GFF/GTF file, FASTA file, custom annotation file). To circumvent this you may provide VEP with a synonyms file. A synonym file is included in VEP's cache files, so if you have one of these for your species you can use it as follows:

```
./vep -i input.vcf -cache -synonyms ~/.vep/homo_sapiens/88_GRCh38/chr_synonyms.txt
```

The file consists of lines containing pairs of tab-separated synonyms. Order is not important as synonyms can be used in both "directions".

Q: Can I get gnomAD allele frequencies in VEP?

Yes, see this guide.

Q: Why do I see the following error?

```
Could not connect to database homo_sapiens_core_63_37 as user anonymous using [DBI:mysql:database=ho
Unknown MySQL server host 'ensembldb.ensembl.org' (2) at $HOME/src/ensembl/modules/Bio/EnsEMBL/DBSQL

-------------------- EXCEPTION --------------------
MSG: Could not connect to database homo_sapiens_core_63_37 as user anonymous using [DBI:mysql:databa
Unknown MySQL server host 'ensembldb.ensembl.org' (2)
```

A: By default the VEP script is configured to connect to the public MySQL server at ensembldb.ensembl.org. Occasionally the server may break connection with your script, which causes this error. This can happen when the server is busy, or due to various network issues. Consider using a local copy of the database, or the caching system.

Q: Can I use the VEP script on Windows?

Yes - see the documentation for a few different ways to get the VEP running on Windows.

Q: Can I download all of the SIFT and/or PolyPhen predictions?

A: The Ensembl Variation database and the human VEP cache file contain precalculated SIFT and PolyPhen predictions for every possible amino acid change in every translated protein product in Ensembl. Since these data are huge, we store them in a compressed format. The best approach to extract them is to use our Perl API.

The format in which the data are stored in our database is described here

The simplest way to access these matrices is to use an API script to fetch a ProteinFunctionPredictionMatrix for your protein of interest and then call its 'get_prediction' method to get the score for a particular position and amino acid, looping over all possible amino acids for your position. There is some detailed documentation on this class in the API documentation here.

You would need to work out which peptide position your codon maps to, but there are methods in the TranscriptVariationAllele class that should help you (probably translation_start and translation_end).