










Use the VEP script 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 the [VEP](#) and Ensembl.

The VEP is a powerful and highly configurable tool - have a browse through the  [documentation](#). You might also like to read up on the  [data formats](#) that the 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 the VEP in your work, please cite **McLaren et. al.** ([doi:10.1093/bioinformatics/btq330](https://doi.org/10.1093/bioinformatics/btq330) )

Any questions? Send an email to the Ensembl [developers' mailing list](#) or contact the [Ensembl Helpdesk](#).

★ Quick start

1. [Download latest version \(81\)](#)

2. Unpack

```
unzip ensembl-tools-release-81.zip
```

3. Install

```
cd ensembl-tools-release-81/scripts/variant_effect_predictor/  
perl INSTALL.pl
```

4. Test

```
perl variant_effect_predictor.pl -i example.vcf --cache
```

Documentation contents

 [Download documentation in PDF format](#)

★ [Getting started](#)

- [Tutorial](#)

[Download and install](#)

- [Download](#)
- [Installation](#)

[Running the VEP](#)

- [Options](#)
- [Performance](#)

[Caches and databases](#)

- [Caches](#)

[Custom annotations](#)

- [Data formats](#)
- [Options](#)

[Plugins](#)

- [Examples](#)

[Other information](#)

- [Multiple assemblies](#)
- [Summarising annotation](#)
- [HGVS notations](#)
- [RefSeq transcripts](#)

- [Using the VEP in Windows](#)

- [Databases](#)

- [Using plugins](#)

- [File conversion](#)

[Data formats](#)

- [Input](#)
- [Output](#)

[Filtering results](#)

- [Running the script](#)
- [Writing filters](#)


[Examples & use cases](#)


- [Example commands](#)
- [Citations and VEP users](#)

[FAQ](#)

- [General questions](#)
- [Web VEP questions](#)
- [VEP script questions](#)

Basic tutorial

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 the VEP yet? Click this link to  [download](#) if you haven't already. If you prefer to do things on the terminal, you can use curl to grab the VEP archive:

```
> curl -LO "https://github.com/Ensembl/ensembl-tools/archive/release/81.zip"
```

If you clicked the download link, let's find the file and unpack it ourselves. Typically the file will be in your **Downloads** folder:

```
> cd ~/Downloads
```

Now let's unpack it and navigate to the VEP's sub-directory:

```
> unzip ensembl-tools-release-81.zip  
> cd ensembl-tools-release-81/scripts/variant_effect_predictor/
```

NOTE: depending on your setup the archive file may instead be named **81.zip**

The 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 v81 of the Ensembl API for use by the 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 v81 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

The 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 spaces):
1 : ailuropoda_melanoleuca_vep_81_ailMell1.tar.gz
2 : anas_platyrhynchos_vep_81_BGI_duck_1.0.tar.gz
3 : anolis_carolinensis_vep_81_Anocar2.0.tar.gz
...
24 : homo_sapiens_vep_81_GRCh38.tar.gz
...
?
```

Type "24" (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 the VEP assumes you are working in human; it's easy to switch to any other species using [--species \[species\]](#).

```
? 24
- downloading ftp://ftp.ensembl.org/pub/release-81/variation/VEP/homo_sapiens_vep_81_GRCh38.tar.gz
- unpacking homo_sapiens_vep_81_GRCh38.tar.gz

Success
```

By default the 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.

The 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). The VEP can also use common formats such as VCF and pileup as input; these are commonly produced by variant calling packages. Have a look at the [Data formats](#) page for more information.

We can now use our cache file to run the VEP on the supplied example file **example_GRCh38.vcf**, which is a VCF file containing variants from the 1000 Genomes Project exon-sequencing pilot, remapped to GRCh38:

```
> perl variant_effect_predictor.pl -i example_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. The VEP tries not to trample over your existing files unless you tell it to. So let's tell it to using [--force_overwrite](#)

```
> perl variant_effect_predictor.pl -i example_GRCh38.vcf --cache --force_overwrite
2013-07-31 09:19:30 - Read existing cache info
2013-07-31 09:19:30 - Starting...
2013-07-31 09:19:30 - Detected format of input file as vcf
2013-07-31 09:19:30 - Read 173 variants into buffer
2013-07-31 09:19:30 - Reading transcript data from cache and/or database
[=====] [ 100% ]
2013-07-31 09:19:31 - Retrieved 3037 transcripts (0 mem, 3106 cached, 0 DB, 69 duplicates)
2013-07-31 09:19:31 - Analyzing chromosome 21
2013-07-31 09:19:31 - Analyzing variants
[=====] [ 100% ]
2013-07-31 09:19:32 - Calculating consequences
[=====] [ 100% ]
2013-07-31 09:19:32 - Analyzing chromosome 22
2013-07-31 09:19:32 - Analyzing variants
[=====] [ 100% ]
2013-07-31 09:19:33 - Calculating consequences
[=====] [ 100% ]
2013-07-31 09:19:34 - Processed 173 total variants (43 vars/sec, 43 vars/sec total)
2013-07-31 09:19:34 - Wrote stats summary to variant_effect_output.txt_summary.html
2013-07-31 09:19:34 - Finished!
```

Note that the VEP has told us it retrieved 3037 transcripts from the cache - this means the cache we downloaded is working OK.

The VEP produces status output as it runs to give you an idea of the progress it is making. You can silence this output using [--quiet](#), though note that errors and warnings will still be printed out.

By default the 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 v81
## Output produced at xxxx-xx-xx 09:19:30
## Connected to homo_sapiens_core_81_38 on ensembl.ensembl.org
## Using cache in /nfs/users/nfs_w/wm2/.vep/homo_sapiens/81
## Using API version 81, DB version 81
## Extra column keys:
## DISTANCE : Shortest distance from variant to transcript
#Uploaded_variation Location Allele Gene Feature Feature_type Consequence ...
rs116645811 21:25587758 A ENSG00000260583 ENST00000567517 Transcript upstream_gene_variant ...
rs116645811 21:25587758 A ENSG00000154719 ENST00000352957 Transcript intron_variant ...
```

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 the VEP's output format, see the [Data formats](#) page.

We can see two lines of output here, both for the uploaded variant named rs116645811. In many cases, a variant will fall in more than one transcript. Typically this is where a single gene has multiple splicing variants; however, as in this case, it can also be that the variant overlaps transcripts from two different genes. Here our variant has a consequence for the genes ENSG00000260583 and ENSG00000154719.

In the consequence column, we can see two different terms, `upstream_gene_variant` and `intron_variant`. These terms are standardised terms 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 the 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. The VEP can give SIFT predictions for any of the missense variants that it predicts. To do this, simply add `--sift b` (the b means we want both the prediction and the score):

```
> perl variant_effect_predictor.pl -i example_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:

```
> perl filter_vep.pl -i variant_effect_output.txt -filter "SIFT is deleterious" | head -n15
...
## SIFT : SIFT prediction
#Uploaded_variation Location Allele Gene Feature ... Extra
rs114053718 21:32656885 G ENSG00000159082 ENST00000382499 ... SIFT=deleterious(0)
rs114053718 21:32656885 G ENSG00000159082 ENST00000382491 ... SIFT=deleterious(0)
rs114053718 21:32656885 G ENSG00000159082 ENST00000322229 ... SIFT=deleterious(0)
rs114053718 21:32656885 G ENSG00000159082 ENST00000433931 ... SIFT=deleterious(0)
```

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 the VEP script. See the [Data formats](#) page for more information on the fields in the Extra column.

You can also configure how the VEP writes its output using the [--fields](#) flag.

You'll also see that we have multiple results for the same gene, ENSG00000159082. 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 the VEP directly into the filtering script:

```
> perl variant_effect_predictor.pl -i example_GRCh38.vcf --cache --force_overwrite --sift b --canonical --symbol \
--fields Uploaded_variation,SYMBOL,CANONICAL,SIFT -o STDOUT | \
perl filter_vep.pl -format vep -filter "CANONICAL is YES and SIFT is deleterious"
...
#Uploaded_variation  SYMBOL      CANONICAL  SIFT
rs114053718         SYNJ1      YES        deleterious(0)
rs2254562           SYNJ1      YES        deleterious(0.04)
rs4986958           IFNGR2     YES        deleterious(0.03)
rs16994704          PIGP       YES        deleterious(0.01)
...
rs115264708         PHF21B     YES        deleterious(0.04)
```

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

Over to you!

This has been just a short introduction to the capabilities of the 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



Download

The Variant Effect Predictor script can be downloaded as a zip from the Ensembl GitHub site:

 [Download latest version \(81\)](#)

It is included as part of the ensembl-tools module of the Ensembl API - you can find it in the ensembl-tools-release-81/scripts/variant_effect_predictor/ directory.

Previous versions: [Show](#)

What's new

New in version 81 (*July 2015*)

- Plugin registry means plugins can be installed from the [VEP installer](#)
- GFF format now supported by VEP's [cache converter](#)
- Fixes and improvements for sequence retrieval from FASTA files

Previous version history: [Show](#)

Requirements

Version 81 of the script requires at least version 81 of the Ensembl Core and Variation APIs and their relevant dependencies to be installed to use the script. A minimal subset of these can be installed using the supplied [installer script](#).

To perform a full install of the API, see the [instructions](#)  for details. To analyse regulatory features, the Ensembl Regulation API should also be installed.

To use the [cache](#), the gzip and zcat utilities are required.

To use the VEP in Windows, see the [instructions for Windows users](#)

Installation

The VEP installer script 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 and configure [cache](#) and [FASTA](#) files for use by the VEP, as well as [plugins](#).

To use the VEP in Windows, see the [instructions for Windows users](#)

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 files. The API set installed by the script is local to the VEP, and will not affect any other Ensembl API installations.

The installer script is also useful for users whose systems do not have all the modules required by the VEP, specifically DBI and DBI::mysql. After configuration using the installer, users can then use the VEP in [offline mode](#) with a cache, eliminating dependency on an Ensembl database (with [limitations](#)).

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), c (cache), f (FASTA), p (plugins) to specify parts to install e.g. "-a ac" for API and cache. You will also need to specify at least one species (see below), and a list of plugins if specified.
--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".
--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: <div>perl -I [dir] variant_effect_predictor.pl</div>
--VERSION [version]	-v	By default the script will install the latest version of the Ensembl API (currently 81). Users can force the script to install a different version at their own risk


<code>--CACHEDIR [dir]</code>	<code>-c</code>	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 <code>--dir</code> flag must be passed when running the VEP if a non-default directory is given: <div>perl variant_effect_predictor.pl --dir [dir]</div>
<code>--UPDATE</code>	<code>-n</code>	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.
<code>--QUIET</code>	<code>-q</code>	Don't write any status output when using <code>--AUTO</code> .
<code>--PREFER_BIN</code>	<code>-p</code>	Use this if the installer fails with out of memory errors.

Using the VEP in Windows

The 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 the VEP script on a Windows machine:


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. Download the Ensembl virtual machine image [\[instructions\]](#)
2. Use a virtual machine in the Amazon Elastic Cloud Service (ECS) [\[Amazon\]](#) 

Cygwin

Cygwin is a collection of tools which provide a Linux environment embedded in Windows. It is very simple to install and set up, and requires the fewest compute resources to create a suitable Linux environment for the VEP. Instructions:


1. Download the [Cygwin setup program](#) 
2. Run the setup program, and click through with the default options until you come to select packages to install
3. Select the following packages to install (all are under the "Perl" node):
 - perl
 - perl-DBD-mysql
 - perl-DBI
4. Click through the rest of the install process

5. Download and unpack the VEP package from the [download link](#)

6. Run the VEP [installer](#)


ActiveState Perl

ActiveState provide a version of Perl available for Windows. This can be used to run the VEP on the Windows MS-DOS command line interface. Instructions:

1. Download and install [ActiveState Perl](#) . 5.14.2 is the recommended version to use.
2. In your Windows Start menu, locate the entry created (it will look something like "ActivePerl 5.14.2 Build 1402"), and click on "Perl Package Manager"
3. In the package manager, click the grey "View all packages" icon in the top-left
4. In the adjacent search box, type "DBI"
5. You should see a package named "DBI" in the list - click the package icon to the left to highlight it and select it for installation
6. You should also see a package named "DBD-mysql" - click the package icon to select it for installation
7. Click the green arrow in the top right of the window to install the packages
8. Download and unpack the VEP package from the [download link](#)
9. Start an MS-DOS window by clicking on the Start Menu, then Run..., type "cmd" in the box and press enter. Change directory to where you unpacked the VEP package using the **cd** command:

```
cd C:\Perl\ensembl-tools\scripts\variant_effect_predictor\
```

10. Run the VEP [installer](#)

11. To use cache files, you will need to download the [gzip utility](#)  and put "gzip.exe" either in your executable path or in the same directory as the variant_effect_predictor.pl script. Add the command line flag '--compress "gzip -dc"' when running the VEP

NB: Under ActiveState Perl on Windows there is a known issue when using cache files including SIFT and/or PolyPhen scores. Users should use either a virtual machine or Cygwin as described above.

Variant Effect Predictor data formats



Input

Both the web and script version of the 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. The VEP can use [VCF](#), [pileup](#) 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 the VEP's output. If not provided, the 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

The 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, the 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 the 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 the 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 the 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 identifier for each of your variants to the VCF file, or use VCF as your output format.

Structural variants

The 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	.		.	.	SVTYPE=DEL;END=1387562	.

See the [VCF definition document](#) for more detail on how to describe structural variants in VCF format.

Pileup

The pileup format can also be used as input for the VEP. This is the output of the ssaha pileup package.

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 permissible 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 output format from the web and script VEP is the same. 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 ";". The keys are as follows:
 - *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
 - *SWISSPROT* - UniProtKB/Swiss-Prot identifier of protein product
 - *TREMBL* - UniProtKB/TrEMBL identifier of protein product
 - *UNIPARC* - UniParc identifier of protein product
 - *HGVSc* - the HGVS coding sequence name
 - *HGVSp* - the HGVS protein sequence name
 - *HGVS_OFFSET* - Indicates by how many bases the HGVS notations for this variant have been [shifted](#)
 - *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 identifier 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 identifier 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
- *GMAF* - Non-reference allele and frequency of existing variant in 1000 Genomes
- *AFR_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined African population
- *AMR_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined American population
- *ASN_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined Asian population
- *EUR_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined European population
- *EAS_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined East Asian population
- *SAS_MAF* - Non-reference allele and frequency of existing variant in 1000 Genomes combined South Asian population
- *AA_MAF* - Non-reference allele and frequency of existing variant in NHLBI-ESP African American population
- *EA_MAF* - Non-reference allele and frequency of existing variant in NHLBI-ESP European American population
- *CLIN_SIG* - Clinical significance of variant from dbSNP
- *BIOTYPE* - Biotype of transcript or regulatory feature
- *TSL* - Transcript support level
- *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
- *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:
 - **rseq_3p_mismatch**: signifies a mismatch between the underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence the model was built from. Specifically, there is a mismatch in the 3' UTR of the RefSeq model.
 - **rseq_5p_mismatch**: signifies a mismatch between the underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence

the model was built from. Specifically, there is a mismatch in the 5' UTR of the RefSeq model.

- **rseq_cds_mismatch:** signifies a mismatch between the underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence the model was built from. Specifically, there is a mismatch in the CDS of the RefSeq model.
- **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 underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence the model was built from (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 underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence the model was built from. 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 underlying genomic sequence of the RefSeq transcript with the corresponding RefSeq mRNA sequence the model was built from. 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 genomic sequence of RefSeq model 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).

Empty values are denoted by '-'. Further fields in the Extra column can be added by [plugins](#) or using [custom annotations](#) in the VEP script. Output fields can be configured using the [--fields](#) flag when running the VEP script.

11_224088_C/A	11:224088	A	ENSG00000142082	ENST00000525319	Transcript	missense_variant	742	716	239
11_224088_C/A	11:224088	A	ENSG00000142082	ENST00000534381	Transcript	5_prime_UTR_variant	-	-	-
11_224088_C/A	11:224088	A	ENSG00000142082	ENST00000529055	Transcript	downstream_variant	-	-	-
11_224585_G/A	11:224585	A	ENSG00000142082	ENST00000529937	Transcript	intron_variant	-	-	-

```
22_16084370_G/A 22:16084370 A - ENSR00000615113 RegulatoryFeature regulatory_region_variant - - -
```

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 v81
## Output produced at 2015-04-01 16:09:38
## Connected to homo_sapiens_core_81_38 on ensembl.ensembl.org
## Using API version 81, DB version 81
## sift version sift5.2.2
## COSMIC version 71
## ESP version 20140509
## gencode version GENCODE 22
## HGMD-PUBLIC version 20142
## genebuild version 2014-07
## regbuild version 13.0
## assembly version GRCh38.p2
## dbSNP version 138
## ClinVar version 201410
## Extra column keys:
## IMPACT : Subjective impact classification
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
```

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 the 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: Allele|Consequence|IMPACT|SY
#CHROM  POS      ID          REF  ALT  QUAL  FILTER  INFO
21      26978790  rs75377686  T    C    .      .      CSQ=C|missense_variant|MODERATE|MRPL39|ENSG00000154719|Transcript|ENST000
```

JSON output

The 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,  
        "hgvs": "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, start, end, strand	The variant's location field is broken down into constituent coordinate parts for clarity. "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

The 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 the VEP writing a stats file, use the flag `--no_stats`. To have the 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]`.

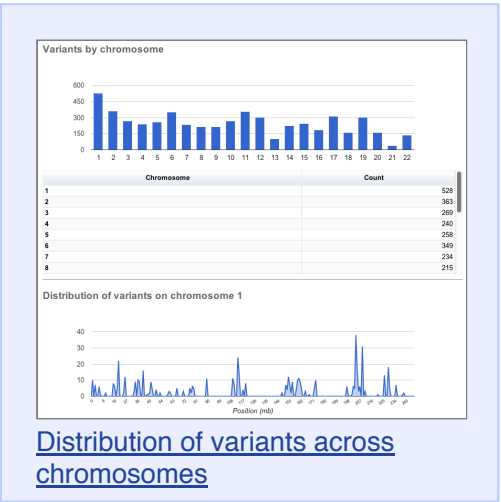
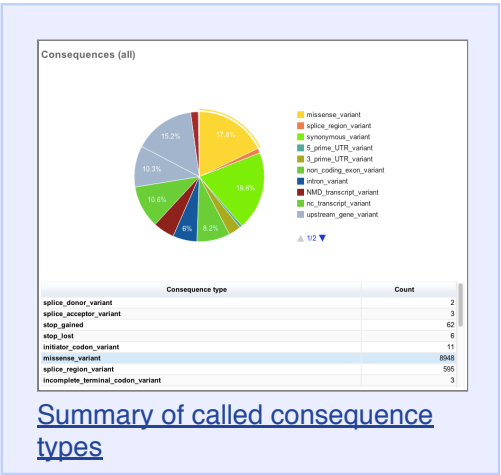
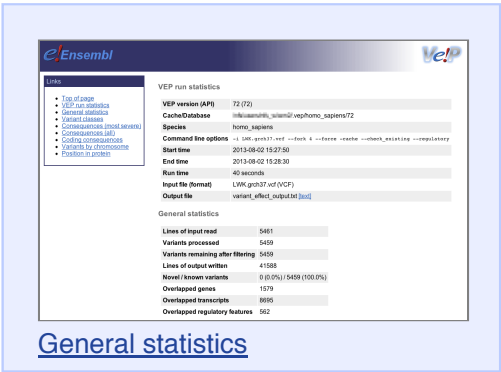
The page contains several sections:

General statistics

This section contains two tables. The first describes the cache and/or database used, the version of the 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.



Variant Effect Predictor Running the VEP



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

```
perl variant_effect_predictor.pl [options]
```

where [options] represent a set of flags and options to the script. These can be listed using the flag [--help](#):

```
perl variant_effect_predictor.pl --help
```

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

For smaller input files, it is possible for the 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:

```
perl variant_effect_predictor.pl --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](#)).

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
--verbose	-v	Output longer status messages as the script runs. This option can be used to generate the basis of a configuration file - see --config below. <i>Not used by default</i>
--quiet	-q	Suppress status and warning messages. <i>Not used by default</i>

<code>--no_progress</code>	Don't show progress bars. <i>Progress bars shown by default</i>
<code>--config [filename]</code>	<p>Load configuration options from a config file. The config file should consist of whitespace-separated pairs of option names and settings e.g.:</p> <pre>output_file my_output.txt species mus_musculus format vcf host useastdb.ensembl.org</pre> <p>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. <i>Not used by default</i></p>
<code>--everything</code>	<p>Shortcut flag to switch on all of the following:</p> <p>--sift b, --polyphen b, --ccds, --uniprot, --hgvs, --symbol, --numbers, --domains, --regulatory, --canonical, --protein, --biotype, --uniprot, --tsl, --gmaf, --maf 1kg, --maf esp, --pubmed, --variant class</p>
<code>--fork [num_forks]</code>	Enable forking , using the specified number of forks. Forking can dramatically improve the runtime of the script. <i>Not used by default</i>

Input options

Flag	Alternate	Description
<code>--species [species]</code>		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. <i>Default = "homo_sapiens"</i>
<code>--assembly [name]</code>		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. <i>Default = use found assembly version</i>
<code>--input_file [filename]</code>	<code>-i</code>	Input file name. If not specified, the script will attempt to read from STDIN.
<code>--format [format]</code>		Input file format - one of "ensembl", "vcf", "pileup", "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, pileup or HGVS format, a list of variant identifiers (e.g. rsIDs from dbSNP), or the output from the VEP (e.g. to add custom annotation to an existing results file using --custom). <i>Auto-detects format by default</i>

<code>--output_file [filename] -o</code>	Output file name. The script can write to STDOUT by specifying STDOUT as the output file name - this will force quiet mode. <i>Default = "variant_effect_output.txt"</i>
<code>--force_overwrite</code> <code>--force</code>	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. <i>Not used by default</i>
<code>--stats_file [filename]</code>	Summary stats file name. This is an HTML file containing a summary of the VEP run - the file name must end ".htm" or ".html". <i>Default = "variant_effect_output.txt_summary.html"</i>
<code>--no_stats</code>	Don't generate a stats file.
<code>--stats_text</code>	Generate a plain text stats file in place of the HTML.
<code>--html</code>	Generate an additional HTML version of the output file containing hyperlinks to Ensembl and other resources. File name of this file is [output_file].html

Cache options

Flag	Alternate	Description
<code>--cache</code>		Enables use of the cache . Add --refseq to use the refseq cache (if installed).
<code>--dir [directory]</code>		Specify the base cache/plugin directory to use. <i>Default = "\$HOME/.vep/"</i>
<code>--dir_cache [directory]</code>		Specify the cache directory to use. <i>Default = "\$HOME/.vep/"</i>
<code>--dir_plugins [directory]</code>		Specify the plugin directory to use. <i>Default = "\$HOME/.vep/"</i>
<code>--offline</code>		Enable offline mode . No database connections will be made, and only a complete cache (either downloaded or built using --build) can be used for this mode. Add --refseq to use the refseq cache (if installed). <i>Not used by default</i>
<code>--fasta [file dir]</code>		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. <i>Not used by default</i>
<code>--cache_version</code>		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 74 and the Ensembl Genomes version 21. <i>Not used by default</i>
<code>--show_cache_info</code>		Show source version information for selected cache and quit

Output options

Flag	Alternate	Description
<code>--variant_class</code>		Output the Sequence Ontology variant class . <i>Not used by default</i>
<code>--sift [p s b]</code>		Species limited SIFT predicts whether an amino acid substitution affects protein function based on sequence homology and the physical properties of amino acids. The VEP can output the prediction term , score or both . <i>Not used by default</i>
<code>--polyphen [p s b]</code>	<code>--poly</code>	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. The VEP can output the prediction term , score or both . The VEP uses the humVar score by default - use --humdiv to retrieve the humDiv score. <i>Not used by default</i>
<code>--humdiv</code>		Human only Retrieve the humDiv PolyPhen prediction instead of the default humVar. <i>Not used by default</i>
<code>--regulatory</code>		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. <i>Not used by default</i>
<code>--cell_type</code>		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 . <i>Not used by default</i>
<code>--custom [filename]</code>		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. <i>Not used by default</i>
<code>--plugin [plugin name]</code>		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 . <i>Not used by default</i>
<code>--individual [all ind list]</code>		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. <i>Not used by default</i>
<code>--phased</code>		Force VCF genotypes to be interpreted as phased. For use with plugins that depend on phased data. <i>Not used by default</i>
<code>--allele_number</code>		Identify allele number from VCF input, where 1 = first ALT allele, 2 = second ALT allele etc. <i>Not used by default</i>

<code>--total_length</code>		Give cDNA, CDS and protein positions as Position/Length. <i>Not used by default</i>
<code>--numbers</code>		Adds affected exon and intron numbering to to output. Format is Number/Total. <i>Not used by default</i>
<code>--domains</code>		Adds names of overlapping protein domains to output. <i>Not used by default</i>
<code>--no_escape</code>		Don't URI escape HGVS strings. <i>Default = escape</i>
<code>--keep_csq</code>		Don't overwrite existing CSQ entry in VCF INFO field . <i>Overwrites by default</i>
<code>--vcf_info_field</code>	<code>[CSQ ANN (other)]</code>	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 . <i>Default: CSQ</i>
<code>--terms</code>	<code>[ensembl so]</code> <code>-t</code>	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. <i>Default = "SO"</i>

Identifiers

Flag	Alternate	Description
<code>--hgvs</code>		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 <code>--cache</code> or <code>--offline</code> you must use a FASTA file and <code>--fasta</code> . <i>Not used by default</i>
<code>--shift_hgvs</code>	<code>[0 1]</code>	Enable or disable 3' shifting of HGVS notations. When enabled, this causes ambiguous insertions or deletions (typically in repetitive 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. <i>Default: 1 (shift)</i>
<code>--protein</code>		Add the Ensembl protein identifier to the output where appropriate. <i>Not used by default</i>
<code>--symbol</code>		Adds the gene symbol (e.g. HGNC) (where available) to the output. <i>Not used by default</i>
<code>--ccds</code>		Adds the CCDS transcript identifier (where available) to the output. <i>Not used by default</i>
<code>--uniprot</code>		Adds identifiers for translated protein products from three UniProt -related databases (SWISSPROT, TREMBL and UniParc) to the output. <i>Not used by default</i>

<code>--tsl</code>	Adds the transcript support level for this transcript to the output. <i>Not used by default</i>
<code>--canonical</code>	Adds a flag indicating if the transcript is the canonical transcript for the gene. <i>Not used by default</i>
<code>--biotype</code>	Adds the biotype of the transcript or regulatory feature. <i>Not used by default</i>
<code>--xref_refseq</code>	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. <i>Not used by default</i>

Co-located variants

Flag	Alternate	Description
<code>--check_existing</code>		Checks for the existence of known variants that are co-located with your input. By default the alleles are not compared - to do so, use --check_alleles . <i>Not used by default</i>
<code>--check_alleles</code>		When checking for existing variants, only report a co-located variant if none of the alleles supplied 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. <i>Not used by default</i>
<code>--check_sv</code>		Checks for the existence of structural variants that overlap your input. Currently requires database access. <i>Not used by default</i>
<code>--gmaf</code>		Add the global minor allele frequency (MAF) from 1000 Genomes Phase 1 data for any existing variant to the output. <i>Not used by default</i>
<code>--maf_1kg</code>		Add allele frequency from continental populations (AFR,AMR,ASN,EUR) of 1000 Genomes Phase 1 to the output. Note the reported allele(s) and frequencies are for the non-reference allele from the original data, not necessarily the alternate allele from user input. Must be used with --cache <i>Not used by default</i>
<code>--maf_esp</code>		Include allele frequency from NHLBI-ESP populations. Note the reported allele(s) and frequencies are for the non-reference allele from the original data, not necessarily the alternate allele from user input. Must be used with --cache <i>Not used by default</i>
<code>--old_maf</code>		For --maf_1kg and --maf_esp report only the frequency (no allele) and convert this frequency so it is always a minor frequency, i.e. < 0.5
<code>--pubmed</code>		Report Pubmed IDs for publications that cite existing variant. Must be used with --cache . <i>Not used by default</i>

<code>--failed</code>	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. <i>Exclude by default</i>
-----------------------	--

Data format options

Flag	Alternate	Description
<code>--vcf</code>		<p>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.</p> <p>If the input format was VCF, the file will remain unchanged save for the addition of the CSQ field (unless using any filtering).</p> <p>Custom data added with --custom are added as separate fields, using the key specified for each data file.</p> <p>Commas in fields are replaced with ampersands (&) to preserve VCF format.</p> <p><i>Not used by default</i></p>
<code>--json</code>		Writes output in JSON format . <i>Not used by default</i>
<code>--gvf</code>		Writes output in GVF format. <i>Not used by default</i>
<code>--fields [list]</code>		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. <i>Not used by default</i>
<code>--convert [format]</code>		Converts the input file to the specified format (one of "ensembl", "vcf", "pileup"). See documentation for more details. Converted output is written to the file specified with --output file . No consequence prediction is carried out. <i>Not used by default</i>
<code>--minimal</code>		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 the 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. <i>Not used by default</i>

Filtering and QC options

NOTE: The filtering options here filter your results **before** they are written to your output file. Using the VEP's [filtering script](#), it is possible to filter your results **after** the 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
--check_ref		Force the script to check the supplied reference allele against the sequence stored in the Ensembl Core database. Lines that do not match are skipped. <i>Not used by default</i>
--coding_only		Only return consequences that fall in the coding regions of transcripts. <i>Not used by default</i>
--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 <i>Not used by default</i>
--no_intergenic		Do not include intergenic consequences in the output. <i>Not used by default</i>
--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. <i>Not used by default</i>
--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. <i>Not used by default</i>
--flag_pick		As per --pick , but adds the PICK flag to the chosen block of consequence data and retains others. <i>Not used by default</i>
--flag_pick_allele		As per --pick_allele , but adds the PICK flag to the chosen block of consequence data and retains others. <i>Not used by default</i>
--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 . <i>Not used by default</i>
--pick_order [c1,c2,c3]		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: <i>canonical,tsl,biotype,ccds,rank,length</i>
--most_severe		Output only the most severe consequence per variation. Transcript-specific columns will be left blank. Consequence ranks are given in this table . <i>Not used by default</i>
--summary		Output only a comma-separated list of all observed consequences per variation. Transcript-specific columns will be left blank. <i>Not used by default</i>
--filter_common		Shortcut flag for the filters below - this will exclude variants that have a co-located existing variant with global MAF > 0.01 (1%). May be modified using any of the following freq_* filters. For human, this can be used in offline mode for the following populations: 1KG_ALL, 1KG_AFR, 1KG_AMR, 1KG_ASN, 1KG_EUR. <i>Not used by default</i>

<code>--check_frequency</code>	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. Using this option requires a database connection - while it can be used with <code>--cache</code> , the database will still be accessed to retrieve frequency data. Frequencies used in filtering are added to the output under the FREQS key in the Extra field. <i>Not used by default</i>																
<code>--freq_pop [pop]</code>	<p>Name of the population to use in frequency filter. This can be the name of the population as it appears on the Ensembl website (suitable for most species), or in the following short form for human. 1000 genomes populations are currently pilot 1 (low coverage).</p> <table> <tr> <th>Example value for <code>--freq_pop</code></th><th>Description</th></tr> <tr> <td>1kg_all</td><td>1000 genomes combined population (global)</td></tr> <tr> <td>1kg_afr</td><td>1000 genomes combined African populations (also amr, asn, eur)</td></tr> <tr> <td>1kg_chb</td><td>1000 genomes CHB population</td></tr> <tr> <td>hapmap_yri</td><td>HapMap YRI population</td></tr> <tr> <td>1kg</td><td>Any 1000 genomes phase 1 population</td></tr> <tr> <td>ceu</td><td>Any of HapMap or 1000 genomes CEU populations</td></tr> <tr> <td>any</td><td>Any HapMap or 1000 genomes population</td></tr> </table>	Example value for <code>--freq_pop</code>	Description	1kg_all	1000 genomes combined population (global)	1kg_afr	1000 genomes combined African populations (also amr, asn, eur)	1kg_chb	1000 genomes CHB population	hapmap_yri	HapMap YRI population	1kg	Any 1000 genomes phase 1 population	ceu	Any of HapMap or 1000 genomes CEU populations	any	Any HapMap or 1000 genomes population
Example value for <code>--freq_pop</code>	Description																
1kg_all	1000 genomes combined population (global)																
1kg_afr	1000 genomes combined African populations (also amr, asn, eur)																
1kg_chb	1000 genomes CHB population																
hapmap_yri	HapMap YRI population																
1kg	Any 1000 genomes phase 1 population																
ceu	Any of HapMap or 1000 genomes CEU populations																
any	Any HapMap or 1000 genomes population																
<code>--freq_freq [freq]</code>	Minor allele frequency to use for filtering. Must be a float value between 0 and 0.5																
<code>--freq_gt_lt [gt lt]</code>	Specify whether the frequency of the co-located variant must be greater than (gt) or less than (lt) the value specified with --freq_freq																
<code>--freq_filter [exclude include]</code>	Specify whether to exclude or include only variants that pass the frequency filter																
<code>--allow_non_variant</code>	When using VCF format as input and output, by default the 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.																

Database options

Flag	Alternate	Description
<code>--database</code>		Enable the VEP to use local or remote databases.
<code>--host [hostname]</code>		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. <i>Default = "ensembl.db.ensembl.org"</i>
--user [username]	-u	Manually define the database username. <i>Default = "anonymous"</i>
--password [password]	--pass	Manually define the database password. <i>Not used by default</i>
--port [number]		Manually define the database port. <i>Default = 5306</i>
--genomes		Override the default connection settings with those for the Ensembl Genomes public MySQL server. Required when using any of the Ensembl Genomes species. <i>Not used by default</i>
--gencode_basic		Limit your analysis to transcripts belonging to the GENCODE basic set. This set has fragmented or problematic transcripts removed.
--refseq		<p>Instead of using the core database, use the otherfeatures database to retrieve transcripts. This database 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)</p> <p>You should also specify this option if you have installed the RefSeq cache in order for the VEP to pick up the alternate cache directory.</p>
--merged		Use the merged Ensembl and RefSeq cache. Consequences are flagged with the SOURCE of each transcript used.
--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
--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
--db_version [number]	--db	Force the script to connect to a specific version of the Ensembl databases. Not recommended as there will usually be conflicts between software and database versions. <i>Not used by default</i>
--registry [filename]		Defining a registry file overwrites other connection settings and uses those found in the specified registry file to connect. <i>Not used by default</i>

Advanced options

Flag	Alternate	Description
--no_whole_genome		Force the script to run in non-whole-genome mode. This was the original default mode for the VEP script, but has now been superceded by whole-genome mode, which is the default. In this mode, variants are analysed one at a

time, with no caching of transcript data. *Not used by default*

`--buffer_size [number]`

Sets the internal buffer size, corresponding to the number of variations 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*

`--write_cache`

Enable writing to the cache. *Not used by default*

`--build [all|list]`

Build a complete cache for the selected species from the database. Either specify a list of chromosomes (see [--chr](#) for how to do this), or use
`--build all`

to build for all top-level chromosomes. **WARNING: Do not use this flag when connected to one of the public databases - please instead download a pre-built cache or build against a local database.** *Not used by default*

`--compress [command]`

By default the VEP uses the utility `zcat` to decompress cached files. On some systems `zcat` may not be installed or may misbehave; by specifying one of [--compress gzcat](#) or [--compress "gzip -dc"](#) you may be able to bypass these problems. *Not used by default*

`--skip_db_check`

ADVANCED Force the script to use a cache built from a different host than specified with `--host`. Only use this if you are sure the two hosts are compatible (e.g. `ensembl.db.ensembl.org` can be considered compatible with `useastdb.ensembl.org` as the data is mirrored between the two). *Not used by default*

`--cache_region_size [size]`

ADVANCED The size in base-pairs of the region covered by one file in the cache. By default this is 1MB, which produces approximately ~500 files maximum per sub-directory in human. Reducing this can reduce the amount of memory and decrease the run-time when you use a cache built this way. Note that you must specify the same `--cache_region_size` when both building/writing to the cache and reading from it. *Not used by default*

The VEP script can use a variety of data sources to retrieve transcript information that is used to predict consequence types.

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

Using the cache

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

Cache files are compressed using the gzip utility. By default zcat is used to decompress the files, although gzcat or gzip itself can be used to decompress also - you must have one of these utilities installed in your path to use the cache. Use [--compress \[command\]](#) to change the default.

Pre-built caches

The easiest solution is to download a pre-built cache for your species; this eliminates the need to connect to the database while the script is running (except when using [certain options](#)). Cache files can either be downloaded and unpacked as described here, or automatically downloaded and configured using the [installer script](#).

Users interested in RefSeq transcripts may download an alternate cache file (eg homo_sapiens_refseq), or a merged file of RefSeq and Ensembl transcripts (eg homo_sapiens_merged); remember to specify [--refseq](#) or [--merged](#) when running the VEP to use the relevant cache.

Cache files for popular species:

- [Human \(*Homo sapiens*\) - GRCh37](#)
- [Human \(*Homo sapiens*\) - GRCh38](#)
- [Mouse \(*Mus musculus*\) - GRCm38](#)
- [Zebrafish \(*Danio rerio*\) - Zv9](#)

Other species:

- [Vertebrates](#)
- [Bacteria](#)
- [Fungi](#)
- [Metazoa](#)

- [Plants](#)
- [Protists](#)

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.

Instructions for use

1. Download the archive file for your species
2. Extract the archive in your cache directory. By default the VEP uses \$HOME/.vep/ as the cache directory, where \$HOME is your UNIX home directory.

```
mv homo_sapiens_vep_81.tar.gz ~/.vep/  
cd ~/.vep/  
tar xzf homo_sapiens_vep_81.tar.gz
```

3. Run the VEP with the [--cache](#) option

Caches for several species, and indeed different Ensembl releases of the same species, can be stored in the same cache base directory. The files are stored in the following directory hierarchy: \$HOME -> .vep -> species -> version -> chromosome

If a pre-built cache does not exist for your species, please contact the [ensembl-dev](#)  mailing list and we will endeavour to add your species to the list of downloads.

It is also possible to build your own [cache from a GTF/GFF file](#) and a FASTA file.

It is possible to use any combination of cache and database; when using the cache, the cache will take preference, with the database being used when the relevant data is not found in the cache.

Building a cache from a GTF or GFF file

For species that don't have a publicly available cache, or even an Ensembl core database, it is possible to build a VEP cache using the gtf2vep.pl script included alongside the main script. This requires a GTF or GFF file and a FASTA file containing the reference sequence for the same species.

The first time you run the script, the Bio::DB::Fasta module will create an index for the FASTA file that allows rapid random access to the sequences corresponding to the transcripts described in the GTF/GFF file; this may take a few minutes to create.

The script is run as follows:

```
perl gtf2vep.pl -i my_species_genes.gtf -f my_species_seq.fa -d 81 -s my_species  
perl variant_effect_predictor.pl -offline -i my_species_variants.vcf -s my_species
```

By default the cache is created in `$HOME/.vep/[species]/[version]/` - to change this root directory, use [--dir](#).

This process takes around 15-20 minutes for human (including the time taken to index the FASTA file). Note that caches created in this way can only be used in [offline mode](#).

Using FASTA files

By pointing the 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 the VEP to retrieve HGVS notations ([--hgvs](#)) and check the reference sequence given in input data ([--check_ref](#)) without accessing a database.

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

To enable this the VEP uses the [Bio::DB::Fasta](#) module. The first time you run the script 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, the 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. 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 the VEP requires that the file be unzipped 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:

```
wget ftp://ftp.ensembl.org/pub/release-81/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
gzip -d Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
perl variant_effect_predictor.pl -i input.vcf --offline --hgvs --fasta Homo_sapiens.GRCh38.dna.primary_assembly.fa
```

Convert with tabix

For users with [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, the VEP should automatically detect the converted cache and use this in place. Note that tabix and bgzip must be installed on your system to use a converted cache.

Limitations of the cache

The cache stores the following information:

- Transcript location, sequence, exons and other attributes
- Gene, protein and HGNC identifiers for each transcript (where applicable)
- Location and alleles of existing variations
- Regulatory regions
- Predictions and scores for SIFT, PolyPhen

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

- Frequency filtering of input ([--check frequency](#)) on populations not included in the cache. The human cache currently includes frequency data for the combined 1000 Genomes phase 1 population (ALL), the continental level populations (AFR, AMR, ASN, EUR), and the two NHLBI-ESP populations (AA, EA). It does **not** contain frequencies for national level (e.g. CEU, YRI) populations.
- HGVS names ([--hgvs](#)) - 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 the script 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
```

Data privacy

When using the public database servers, the VEP script 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 suitable for those with sensitive or private data. Users should note that **only** the coordinates are transmitted to the server; no other information is sent.

By using a full downloaded cache (preferably in [offline mode](#)) or a local database, it is possible to avoid completely any network connections to public servers, thus preserving absolutely the privacy of your data.

Offline mode

It is possible to run the VEP in a offline mode that does not use the database, and does not require a standard installation of the Ensembl API. This means users require only

perl (version 5.8 or greater) and the either zcat, gzcata or gzip utilities. To enable this mode, use the flag [--offline](#).

The simplest way to set up your system is to use the [installer script](#), INSTALL.pl. This will download the required dependencies to your system, and download and set up any cache files that you require.

The [limitations](#) described above apply absolutely when using offline mode. For example, if you specify [--offline](#) and [--check_sy](#), the script will report an error and refuse to run.

All other features, including the ability to use [custom annotations](#) and [plugins](#), are accessible in offline mode.

Public database servers

By default, the script is configured to connect to Ensembl's public MySQL instance at [ensembl.mysql.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 the script 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 the script (this can be the same machine). For most of the functionality of the VEP, you will only need the Core database (e.g. `homo_sapiens_core_81_38`) installed. In order to find co-located variations or to use SIFT or PolyPhen, it is also necessary to install the relevant variation database (e.g. `homo_sapiens_variation_81_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'      => 'ensembl.ensembl.org',
'-user'      => 'anonymous',
'-pass'      => '',
'-dbname'    => 'homo_sapiens_core_81_38'
);

Bio::Ensembl::Variation::DBSQL::DBAdaptor->new(
  '-species' => "Homo_sapiens",
  '-group'   => "variation",
  '-port'    => 5306,
  '-host'    => 'ensembl.ensembl.org',
  '-user'    => 'anonymous',
  '-pass'    => '',
  '-dbname'  => 'homo_sapiens_variation_81_38'
);

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

```

For more information on the registry and registry files, see [here](#).

Building your own cache

It is possible to build your own cache using the VEP script. **You should NOT use this command when connected to the public MySQL instances** - the process takes a long time, meaning the connection can break unexpectedly and you will be violating Ensembl's reasonable use policy on the public servers. You should either download one of the pre-built caches, or create a [local](#) copy of your database of interest to build the cache from.

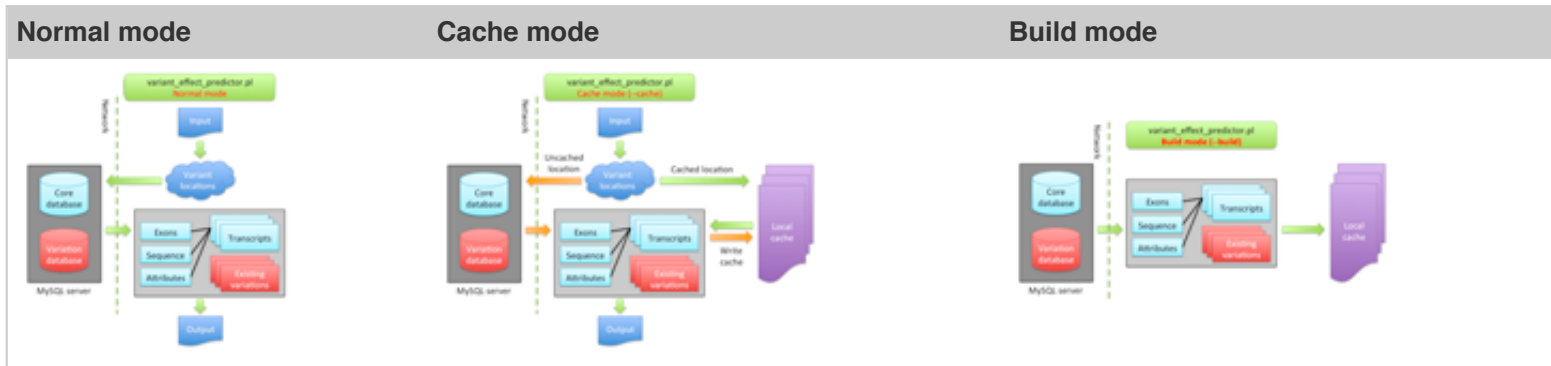
You may wish to build a full cache if you have a custom Ensembl database with data not found on the public servers, or you may wish to create a minimal cache covering only a certain set of chromosome regions. Cache files are compressed using the gzip utility; this must be installed in your path to write cache files.

To build a cache "on-the-fly", use the [--cache](#) and [--write_cache](#) flags when you run the VEP with your input. Only cache files overlapping your input variants will be created; the next time you run the script with this cache, the data will be read from the cache instead of the database. Any data not found in the cache will be read from the database (and then written to the cache if [--write_cache](#) is enabled). If your data covers a relatively small proportion of your genome of interest (for example, a few genes of interest), it can be OK to use the public MySQL servers when building a partial cache.

```
perl variant_effect_predictor.pl -cache -dir /my/cache/dir/ -write_cache -i input.txt
```

To build a cache from scratch, use the flag [--build_all](#) or e.g. [--build 1-5,X](#) to build just a subset of chromosomes. You do not need to specify any of the usual input options when building a cache:

```
perl variant_effect_predictor.pl -host dbhost -user username -pass password -port 3306 -build 21 -dir /my/cache/dir/
```



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 the 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 the 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.

```
$protein_function_prediction_matrix_adaptor->fetch_by_analysis_translation_md5('sift', md5_hex($transcript->{_variation_effec
```

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

Variant Effect Predictor Filtering results



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

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

Running the script

Run the script as follows:

```
perl variant_effect_predictor.pl -i in.vcf -o out.txt -cache -everything
perl filter_vep.pl -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:

```
perl variant_effect_predictor.pl -i in.vcf -o stdout -cache -check_existing | perl filter_vep.pl -filter "not Existing_variation"
```

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

<code>--filter [filters]</code>	<code>-f</code>	Add filter (see below). Multiple <code>--filter</code> flags may be used, and are treated as logical ANDs, i.e. all filters must pass for a line to be printed
<code>--list</code>	<code>-l</code>	List allowed fields from the input file
<code>--count</code>	<code>-c</code>	Print only a count of matched lines
<code>--only_matched</code>		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 <code>--only_matched</code> 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.
<code>--ontology</code>	<code>-y</code>	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 <code>ensembl.db.ensembl.org</code> . Use <code>-host</code> , <code>--port</code> , <code>--user</code> , <code>--password</code> , <code>--version</code> as per <code>variant_effect_predictor.pl</code> 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`
2. **Operator** : The operator defines the comparison carried out.
3. **Value** : The value to which the content of the field is compared.

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 it is defined; in this case the operator and value can be left out:

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

Filter strings can also 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 MAF is greater than 10% in either AFR or ASN populations
--filter "AFR_MAF > 0.1 or ASN_MAF > 0.1"

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

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 the 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"
```

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

```
# 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

- **>=** (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"
```

The VEP script 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. Users should note that the VEP will only look for overlaps (both exact and inexact) with these annotations; for example, any sequence in a GTF file will not be taken into account.

Annotations 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

The VEP supports the following formats:

- [BED](#) : a simple tab-delimited format containing 3-12 columns of data. The first 3 columns contain the coordinates of the feature. If available, the VEP will use the 4th column of the file as the identifier of the feature.
- [GFF](#) : a format for describing genes and other features. If available, the VEP will use the "ID" field as the identifier of this feature.
- [GTF](#) : treated in an identical manner to GFF.
- [VCF](#) : a format used to describe genomic variants. The VEP will use the 3rd column of the file as the identifier.
- [bigWig](#) : a format for storage of dense continuous data. The 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.

Any other files can be easily converted to be compatible with the 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 the 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 the 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:

```
perl variant_effect_predictor.pl -custom myFeatures.gff.gz,myFeatures,gff,overlap,0
perl variant_effect_predictor.pl -custom frequencies.bw,Frequency,bigwig,exact,0
perl variant_effect_predictor.pl -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, the 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, the 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.
- **Force report coordinates** : One of "0" or "1" (if left blank assumed to be "0") - if set to "1", this forces the 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), the 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. Only applies when using VCF format custom files.

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 the VEP is run. From this point on, only the portions of data requested by the script (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. Furthermore, when using bigWig files, the VEP generates temporary files that by default are written to the /tmp/ directory - to override this, use the [--tmpdir /my/tmp/dir](#) flag.

Annotating existing results

It is possible to add custom annotation to existing VEP results files. To do this, you need to specify the [--no_consequence](#) option, and provide your VEP output file as the input file for the script. The script should auto-detect the format of the file; if it does not, you can force it to read the file as VEP output using [--format vep](#).

Variant Effect Predictor Plugins



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

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

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

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-variation/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 and we intend to create a central portal for VEP plugins and other scripts written using Ensembl resources in the near future. In the mean time, please contact the developers mailing list if you want to share your plugin.

If you have VEP plugins or other code to share with the community, [Ensembl elcode](#) is a directory for extensions to Ensembl.

How it works

Plugins are run once the 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 the 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 the 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 does not want to add any data to a particular line it should return an empty hashref. If a plugin instead wants to 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 the 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:

```
perl variant_effect_predictor.pl -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.:

```
perl variant_effect_predictor.pl -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, the 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

```
perl variant_effect_predictor.pl -cache -o stdout
```

- Add regulatory region consequences

```
perl variant_effect_predictor.pl -cache -i variants.txt -regulatory
```

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

```
perl variant_effect_predictor.pl -cache -i variants.vcf.txt -format vcf -symbol
```

- Filter out common variants based on 1000 genomes data

```
perl variant_effect_predictor.pl -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

```
perl variant_effect_predictor.pl -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

```
perl variant_effect_predictor.pl -database -i variants.txt -registry ensembl.registry -sift b -polyphen p
```

- Connect to Ensembl Genomes db server for A.thaliana

```
perl variant_effect_predictor.pl -database -i variants.txt -genomes -species arabidopsis_thaliana
```

- Load config from ini file, run in quiet mode

```
perl variant_effect_predictor.pl -config vep.ini -i variants.txt -q
```

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


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

- Convert RefSeq-based HGVS notations to genomic coordinates in VCF format

```
perl variant_effect_predictor.pl -database -i hgvs.txt -o hgvs.vcf -refseq -convert vcf
```

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

```
perl variant_effect_predictor.pl -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

```
perl variant_effect_predictor.pl -cache -i variants.vcf -plugin MyPlugin -fields Uploaded_variation,Feature,Consequence,MyP
```

1000 Genomes Phase 3 data

1000 Genomes Phase 3 data is available from release 79 in GRCh37 cache files; simply use [--gmaf](#) and/or [--maf 1kg](#) to retrieve frequency data. Due to delays in the build process, this data is not available in release 79 for GRCh38, but there is a way to access the data from the VEP using [custom annotations](#):

1. Ensure your system is set up to access data from tabix-indexed VCF files (see [documentation](#))
2. Download this remapped VCF file and tabix index from Ensembl's FTP server:
 - [VCF](#)
 - [tabix index](#)
3. Run VEP with the following command (using the GRCh38 input example) to get locations and continental-level allele frequencies:

```
perl variant_effect_predictor.pl -i example_GRCh38.vcf -cache \  
-custom 1KG.phase3.GRCh38.vcf.gz,P3,vcf,exact,0,AF,AFR_AF,AMR_AF,EAS_AF,EUR_AF,SAS_AF
```

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

```
## P3 : 1KG.phase3.GRCh38.vcf.gz (exact)  
## P3_AF : AF field from 1KG.phase3.GRCh38.vcf.gz  
## P3_AFR_AF : AFR_AF field from 1KG.phase3.GRCh38.vcf.gz  
## P3_AMR_AF : AMR_AF field from 1KG.phase3.GRCh38.vcf.gz  
## P3_EAS_AF : EAS_AF field from 1KG.phase3.GRCh38.vcf.gz  
## P3_EUR_AF : EUR_AF field from 1KG.phase3.GRCh38.vcf.gz
```

```
## P3_SAS_AF : SAS_AF field from 1KG.phase3.GRCh38.vcf.gz
```

where the P3 field contains the ID (or coordinates if no ID found) of the variant in the VCF file

GERP / conservation scores

You can use the 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:

```
perl variant_effect_predictor.pl -cache -i example.vcf -custom All_hg19_RS.bw,GERP,bigwig
```

Note that to make use of bigWig files you will need the bigWigToWig utility from the [Kent source package](#) installed in your path.

Example conservation score files:

- [GERP Human \(GRCh37\)](#)
- [phastCons Human \(GRCh37\)](#)
- [phyloP Human \(GRCh37\)](#)

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 dbNSFP2.0.zip
cat dbNSFP2.0_variant.chr* > dbNSFP
rm dbNSFP2.0_variant.chr*
bgzip dbNSFP
tabix -s 1 -b 2 -e 2 dbNSFP.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 the VEP with the plugin, you will need to select some of the columns that you wish to retrieve; to list them run the VEP with the plugin and the path to the dbNSFP file and no further parameters:

```
perl variant_effect_predictor.pl -cache -force -plugin dbNSFP,dbNSFP.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. Available columns are:
#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,
aapos,SIFT_score,Polyphen2_HDIV_score,Polyphen2_HDIV_pred,Polyphen2_HVAR_score,
Polyphen2_HVAR_pred,LRT_score,LRT_pred,MutationTaster_score,MutationTaster_pred,
MutationAssessor_score,MutationAssessor_pred,FATHMM_score,GERP++_NR,GERP++_RS,
phyloP,29way_pi,29way_logOdds,LRT_Omega,UniSNP_ids,1000Gp1_AC,1000Gp1_AF,
1000Gp1_AFR_AC,1000Gp1_AFR_AF,1000Gp1_EUR_AC,1000Gp1_EUR_AF,1000Gp1_AMR_AC,
1000Gp1_AMR_AF,1000Gp1_ASN_AC,1000Gp1_ASN_AF,ESP6500_AA_AF,ESP6500_EA_AF
[Ctrl-C]
```

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:

```
perl variant_effect_predictor.pl -cache -force -plugin dbNSFP,dbNSFP.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 the VEP in the latest release may differ from the sequence used to calculate the scores in dbNSFP.

Citations and VEP users

The 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 the 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 the VEP to AnnoVar
- [Pabinger *et al*](#)  reviews variant analysis software, including the VEP

Getting VEP to run faster

Set up correctly, the VEP script is capable of processing around 3 million variants in 1 hour. 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 the VEP and 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 the 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 the 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. Download a [FASTA file](#) if you use [--hgvs](#) or [--check_ref](#). Again, this will prevent the VEP accessing the database unnecessarily (in this case to retrieve genomic sequence).
4. Using forking enables the 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 the 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:

```
perl variant_effect_predictor.pl -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.

5. If you use [--check_existing](#) or any flags that invoke it (e.g. [--gmaf](#), [--maf 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.
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 [Ensembl::XS](#) package. This contains compiled versions of certain key subroutines used in the VEP that will run faster than the default native Perl equivalents. Using this should improve runtime by 5-10%.
9. The VEP is optimised to run on input files that are sorted in chromosomal order. Unsorted files will still work, albeit more slowly.
10. For very large files (for example those from whole-genome sequencing), the VEP process can be easily parallelised by dividing your file into chunks (e.g. by chromosome). The 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 | perl variant_effect_predictor.pl -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 (81) 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 the VEP script.

Once you have installed the relevant cache and FASTA file, you are then able to use the 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 match
```

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

For users looking to move their data between assemblies, Ensembl provides an assembly converter tool - if you've downloaded the 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 the 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 the 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](#) or [--flag pick allele](#)) and to post-filter results.

- [--pick](#): this is the option we anticipate will be of use to most users. The 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. [transcript support level](#)
 3. biotype of transcript (protein_coding preferred)
 4. CCDS status of transcript
 5. consequence rank according to [this table](#)
 6. transcript or feature length (longer preferred)
- [--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
- [--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 the VEP's [filtering script](#)
- [--flag pick allele](#): as above, but flags one block per allele
- [--per gene](#): as [--pick](#), but chooses one annotation block per gene that the input variant overlaps
- [--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

The VEP script supports using HGVS notations as input. This feature is currently under development, and not all HGVS notation types are supported. Specifically, only notations relative to genomic (g) or coding (c) sequences are currently 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 the VEP script 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 the VEP can use RefSeq transcripts.

RefSeq transcripts

Ensembl produces Core schema databases containing alignments of RefSeq transcript objects to the reference genome. This is the [otherfeatures database](#), and is produced for human and mouse. The database also contains alignments of CCDS transcripts and Ensembl EST sequences - they may be included in your analysis using [--all_refseq](#). By passing the [--refseq](#) flag when running the VEP script, 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 should note that RefSeq sequences may disagree with the reference sequence to which they are aligned, hence results generated when using this option should be interpreted with a degree of caution. A much more complex and stringent process is used to produce the main Ensembl Core database, and this should be used in preference to the RefSeq transcripts.

SIFT and PolyPhen predictions and scores are now calculated and referred to internally using the translated sequence, so predictions are available using the [--refseq](#) flag where the RefSeq translation matches the Ensembl translation (they will match in the vast majority of cases - most differences between Ensembl and RefSeq transcripts occur in non-coding regions).

File conversion

The VEP script can be used to [convert](#) files between the various formats that it parses. This may be useful for a user with, for example, a number of variants given in HGVS notation against RefSeq transcript identifiers. The conversion process allows these notations to be converted into genomic reference coordinates, and then used to predict consequences in the VEP against Ensembl transcripts.

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 the 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 the VEP run faster?

There are a number of factors that influence how fast the 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/Variation/Uutils/Sequence.pm line 51
Use of uninitialized value $ref_allele in string eq at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/Ensembl/Variation/Uutils/Sequence.pm line 51
Use of uninitialized value in concatenation (.) or string at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/Ensembl/Variation/Uutils/Sequence.pm line 51
```

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: Can I get 1000 Genomes Phase 3 allele frequencies using GRCh38?

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=homo_sapiens_core_63_37;host=
```

```
Unknown MySQL server host 'ensembladb.ensembl.org' (2) at $HOME/src/ensembl/modules/Bio/Ensembl/DBSQL/DBConnection.pm line 290.  
  
----- EXCEPTION -----  
MSG: Could not connect to database homo_sapiens_core_63_37 as user anonymous using [DBI:mysql:database=homo_sapiens_core_63_37;  
Unknown MySQL server host 'ensembladb.ensembl.org' (2)
```

A: By default the VEP script is configured to connect to the public MySQL server at ensembladb.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).