Variant Effect Predictor — Command line VEP

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

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 VEP uses, and the different ways you can access genome data. The VEP script can annotate your variants with custom data, be extended with plugins, and use powerful filtering to find biologically interesting results.

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

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

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

Quick start

1. Download

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

2. Install

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

3. Test

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

Documentation contents

Download documentation in PDF format

Install VEP

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

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

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

```
perl INSTALL.pl

Hello! This installer is configured to install v103 of the Ensembl API for use by VEP.
It will not affect any existing installations of the Ensembl API that you may have.

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

Checking for installed versions of the Ensembl API...done
It looks like you already have v103 of the API installed.
You shouldn't need to install the API

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

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

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

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

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

Downloading list of available cache files
The following species/files are available; which do you want (can specify multiple separated by spaces):
1 : ailuropoda_melanoleuca_vep_103_ailMel1.tar.gz
2 : anas_platyrhynchos_vep_103_BGI_duck_1.0.tar.gz
3 : anolis_carolinensis_vep_103_AnoCar2.0.tar.gz
...
42 : homo_sapiens_vep_103_GRCh38.tar.gz
...

?
```

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

```
? 42
 - downloading /release-103/variation/vep/homo_sapiens_vep_103_GRCh38.tar.gz
 - unpacking homo_sapiens_vep_103_GRCh38.tar.gz

Success
```

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

Run VEP

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

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

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

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

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

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

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

```
head variant_effect_output.txt

## ENSEMBL VARIANT EFFECT PREDICTOR v103.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_103_38 on ensembldb.ensembl.org
```

```
## Using cache in /homes/user/.vep/homo_sapiens/103_GRCh38
## Using API version 103, DB version 103
## polyphen version 2.2.2
## sift version sift5.2.2
## COSMIC version 78
## ESP version 20141103
## gencode version GENCODE 25
## genebuild version 2014-07
## HGMD-PUBLIC version 20162
## regbuild version 16
## assembly version GRCh38.p7
## ClinVar version 201610
## dbSNP version 147
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Feature_type : Type of feature - Transcript, RegulatoryFeature or MotifFeature
## Consequence : Consequence type
## cDNA_position : Relative position of base pair in cDNA sequence
## CDS_position : Relative position of base pair in coding sequence
## Protein_position : Relative position of amino acid in protein
## Amino_acids : Reference and variant amino acids
## Codons : Reference and variant codon sequence
## Existing_variation : Identifier(s) of co-located known variants
## Extra column keys:
## IMPACT : Subjective impact classification of consequence type
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
## FLAGS : Transcript quality flags
#Uploaded_variation  Location     Allele  Gene             Feature          Feature_type  Consequence         ...
rs7289170           22:17181903  G       ENSG00000093072  ENST00000262607  Transcript    synonymous_variant ...
rs7289170           22:17181903  G       ENSG00000093072  ENST00000330232  Transcript    synonymous_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 VEP's output format, see the Data formats page.

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

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

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

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

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

```
./filter_vep -i variant_effect_output.txt -filter "SIFT is deleterious" | grep -v "##" | head -n5

#Uploaded_variation  Location     Allele  Gene             Feature          ...  Extra
rs2231495           22:17188416  C       ENSG00000093072  ENST00000262607  ...  SIFT=deleterious(0.05)
rs2231495           22:17188416  C       ENSG00000093072  ENST00000399837  ...  SIFT=deleterious(0.05)
rs2231495           22:17188416  C       ENSG00000093072  ENST00000399839  ...  SIFT=deleterious(0.05)
rs115736959         22:19973143  A       ENSG00000099889  ENST00000263207  ...  SIFT=deleterious(0.01)
```

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

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

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

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --sift b --canonical --symbol --tab --fields Uploaded_variation,SYM
./filter_vep --filter "CANONICAL is YES and SIFT is deleterious"

...

#Uploaded_variation  SYMBOL  CANONICAL  SIFT
rs2231495           CECR1   YES        deleterious(0.05)
rs115736959         ARVCF   YES        deleterious(0.01)
rs116398106         ARVCF   YES        deleterious(0)
rs116782322         ARVCF   YES        deleterious(0)
...                 ...     ...        ...
rs115264708         PHF21B  YES        deleterious(0.03)
```

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

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

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --af_1kg -o STDOUT | grep -v "##" | head -n2

#Uploaded_variation  Location     Allele  Gene             Feature          ...  Existing_variation  Extra
rs7289170           22:17181903  G       ENSG00000093072  ENST00000262607  ...  rs7289170           IMPACT=LOW;STRAND=-1;AFR_AF=0.2390
```

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

Over to you!

This has been just a short introduction to the capabilities of VEP - have a look through some more of the options, see them all on the command line using --help, or try using the shortcut --everything which switches on almost all available output fields! Try out the different options in the filtering tool, 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

Download ensembl-vep package (see below the different ways to download it) and then follow the installation instructions.

Using Git

- **Clone the Git repository**

Use git to download the ensembl-vep package:

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

- **Update to a newer version**

To update from a previous version:

```
cd ensembl-vep
git pull
git checkout release/103
perl INSTALL.pl
```

- **Use an older version**

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

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

Download the Zipped package file

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

```
curl -L -O https://github.com/Ensembl/ensembl-vep/archive/release/103.zip
unzip 103.zip
cd ensembl-vep-release-103/
```

Previous versions (ensembl-tools)

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

- Download version 87 (Ensembl 87)
- Download version 86 (Ensembl 86)
- Download version 85 (Ensembl 85)
- Download version 84 (Ensembl 84)
- Download version 83 (Ensembl 83)
- Download version 82 (Ensembl 82)
- Download version 81 (Ensembl 81)
- Download version 80 (Ensembl 80)
- Download version 79 (Ensembl 79)
- Download version 78 (Ensembl 78)
- Download version 77 (Ensembl 77)
- Download version 76 (Ensembl 76)
- Download version 75 (Ensembl 75)
- Download version 74 (Ensembl 74)
- Download version 73 (Ensembl 73)
- Download version 72 (Ensembl 72)
- Download version 71 (Ensembl 71)
- Download version 2.8 (Ensembl 70)
- Download version 2.7 (Ensembl 69)
- Download version 2.6 (Ensembl 68)
- Download version 2.5 (Ensembl 67)
- Download version 2.4 (Ensembl 66)
- Download version 2.3 (Ensembl 65)
- Download version 2.2 (Ensembl 64 - ensembl-tools/scripts/variant_effect_predictor)
- Download version 2.1 (Ensembl 63)
- Download version 2.0 (Ensembl 62 - ensembl-variation/scripts/examples)

What's new?

New in version 103 ***(August 2020)***

- **New:** Variant Recoder is now available as a web tool
- Variant Recoder output is now allele specific
- Web VEP Options:
  - Variant Synonyms are now available through the web interface
  - MasterMind results are available through the REST and web interfaces
- VEP Options:
  - --mane: Now provides additional Mane Plus Clinical annotations

Previous version history - from version 88:

New in version 102 ***(November 2020)***

- VEP options:
  - --uniprot: Now we report precise Ensembl translation to UniProt isoform mappings.
  - --spdi - **new**: Add genomic SPDI notation.
- Web VEP options:
  - Shifting variants in the 3' direction with --shift_3prime and --shift_genomic is now supported through the web interface.
  - DisGeNET - **new**: DisGeNET results are available through the web interface.
  - SpliceAI - **new**: SpliceAI pre-calculated scores are available through the web interface.
- VEP filter options:
  - --soft_filter - **new**: Option to only flag the failing variation in the FILTER column and keep the entries in the output VCF file.

New in version version 101 ***(Aug 2020)***

- New options:
  - --var_synonyms: Report known synonyms for colocated variants. Must be used with --cache.
- VEP plugins:
  - neXtProt - **new**: neXtProt retrieves comprehensive human-centric protein-related data for missense variants

New in version 100 ***(Apr 2020)***

- Human GRCh37 variant and phenotype data has been updated with multiple data sets including dbSNP153, ClinVar's 201912 release and COSMIC release 90
- The GRCh37 RefSeq transcript set has been updated to NCBI's 1st November 2019 release (initially annotated on GCF_000001405.25)!
- New options:
  - --shift_3prime: Right aligns all variants relative to their associated transcripts prior to consequence calculation
  - --shift_genomic: Right aligns all variants, including intergenic variants, before consequence calculation and updates the *Location* field
- VEP plugins:
  - SpliceAI - **new**: SpliceAI is a deep neural network, developed by Illumina, Inc that predicts splice junctions from an arbitrary pre-mRNA transcript sequence.
  - DisGeNET - **new**: DisGeNET is a database containing human variant-disease associations

New in version 99 ***(Jan 2020)***

- Human GRCh38 cache files now contain variants from dbSNP153
- New options have been added to REST:
  - vcf_string: VEP can now provide a VCF-like string representing the input variant
  - transcript_version: Add version numbers to Ensembl transcript identifiers
  - SpliceRegion: Provides granular predictions of splicing effects (Details)
  - LoF: LOFTEE implements a set of filters to predict LoF (loss-of-function) variants. (Details)

New in version 98 ***(Sept 2019)***

- Human GRCh38 cache files now contain variants from dbSNP152
- This employs a new clustering strategy which may result in different rsIDs being reported as known variants for some insertions and deletions - for more information see here
- --clin_sig_allele has been updated to be used by default
- New options:
  - --custom_multi_allelic: prevents VEP from assuming that comma separated lists in custom annotations are allele specific
- MANE attributes are now included within VEP cache files, web VEP and REST
- VEP plugins:
  - satMutMPRA - **new**: measures variant effects on gene RNA expression for 21 regulatory elements
- VEP Installer:
  - HTSLib v1.9 is now installed by default (previously v1.3.2)
  - Bio::DB::HTS v2.11 is now installed by default (previously v2.9)
  - New option 'PLUGINSDIR' allows you to specify the installation directory for plugins

New in version 97 ***(July 2019)***

- Allele-specific clinical significance reported (it was previously variant-specific).
- New options:

  - --clin_sig_allele: report allele specific clinical significance.
  - --mane: report if a transcript is the MANE Select.
  - --max_sv_size: extend the maximum Structural Variant size VEP can process.
  - --no_check_variants_order: permit the use of unsorted input files (WARNING - this is slow and requires more memory).
  - --overlaps: report the proportion and length of a transcript overlapped by a structural variant in VCF format.
- Include the --mane option into the --everything group option.
- Update --pick and --pick_order to support MANE Select transcripts.
- Check if the input variants are ordered: non ordered variants slow down VEP and require more memory.
- Skip annotation of complex and long structural variants and display a warning message.
- Variant recoder: add an option --vcf_string to return results in VCF format.
- VEP plugins:
  - FunMotifs - **new**: provide information about overlapping tissue-specific transcription factor motifs.
  - Mastermind - **new**: reports variants that have clinical evidence cited in the medical literature.
  - StructuralVariantOverlap - **new**: provide information from overlapping structural variants.
  - G2P - **update**: now the plugin can be run offline.
  - Phenotypes - **update**: change the format of the data file (from BED to GVF).
- VEP web tool: the transcript identifiers are now returned with versions unless otherwise specified.
- VEP installer: tabix-indexed variant cache files are now installed by default.

New in version 96 ***(April 2019)***

- Add **SPDI** format for VEP (input) and Variant Recoder (input and output).
- Update VEP cache with **gnomAD 2.1** (human).
- Update the Docker VEP base image to **Ubuntu 18.04**.
- Retire deprecated flags: --gmaf, --maf_1kg, --maf_esp, --maf_exac, --check_alleles, --html, --gvf.
- Retire legacy code about the pileup input format, which is no longer supported.
- Deprecate the installation flag "--VERSION"
- Force numbers to be encoded as numbers in JSON output
- VEP plugins:
  - NearestExonJB - **new**: find the nearest exon junction boundary to a coding sequence variant.
  - Conservation - update: can use BigWig files instead of the Ensembl Compara database.
  - dbNSFP - update: support of the dbNSFP data version 4.
  - Phenotypes - update: possibility to report the phenotype description(s) and other information.
  - PostGAP - update: replace the plugin name POSTGAP to PostGAP.

New in version 95 ***(January 2019)***

- The VEP parser is now more permissive for the GFF files (ID attribute only required for genes and transcripts)
- Add new option --show_ref_allele to include the allele reference in the VEP default output and the tab output formats
- Add a warning message when the VEP annotations INFO field hasn't been found/recognised in the VCF input file
- VEP Docker image:
  - Reduce the size of the VEP Docker image by about 45%.
  - Include the Linkage disequilibrium script in the VEP Docker image, making possible to run the LD plugin
- New VEP plugins:
  - Reference quality
  - OpenTargets results (POSTGAP)
  - Single letter amino acid for HGVS

New in version 94 ***(October 2018)***

- RefSeq transcript version updated.
- Minor updates on the VEP web tool interface.
- When the input data format is not specified on the command line, VEP attempts to detect it. The assumed format is now reported in verbose mode (--verbose).
- VEP assigns assigned the consequence types *TF_binding_site_variant*, *TFBS_ablation*, *TFBS_fusion*, *TFBS_amplification* and *TFBS_translocation* to human and mouse variants which overlapped motif features. These annotations will not be available in VEP caches for human in release 94 so must be added as a custom annotation.

New in version 93 ***(July 2018)***

- Update the JSON output format (allele frequencies) for the Ensembl REST - VEP endpoints. See more information.
- The new Ensembl release brings more frequency data from gnomAD.
- Add the possibility to print the content of the FILTER column (from the VCF custom annotation files) in the output.
- Include the Ensembl/ensembl-xs repository in Docker image to speed up the VEP container.
- Add a new consequence 'extended_intronic_splice_region_variant' in the SpliceRegion VEP plugin.

New in version 92 ***(April 2018)***

- New VEP plugin REVEL (see REVEL plugin).
- Get ambiguity code with --ambiguity.
- GFF/GTF files with exons assigned to multiple transcripts are now supported.

- Improved 1000 Genomes Project frequencies.

New in version 91 ***(December 2017)***

- New input format "region" allows REST-style input to VEP.
- Replace your input variant reference allele with the correct one from the genome with --lookup_ref.
- Add version numbers to Ensembl transcripts with --transcript_version.

New in version 90 ***(August 2017)***

- gnomAD exomes allele frequencies now available with --af_gnomad, replacing ExAC. gnomAD genomes and ExAC are available via custom annotation.
- VEP is now available as a Docker image.
- RefSeq transcripts in VEP cache files are now "corrected" from the reference genome sequence.
- VEP's algorithm for matching colocated known variants has been overhauled - details.
- Change VEP's default (5kb) up/downstream distance with --distance. This supercedes the functionality of the UpDownDistance VEP plugin.
- Feed input directly to VEP with --input_data.
- Suppress header output with --no_headers.
- Detailed installation instructions for Bio::DB::BigFile to access bigWig custom annotation files.

New in version 89 ***(May 2017)***

- exclude known variants with unknown (null) alleles with --exclude_null_alleles.
- write compressed output with --compress_output.
- improved matching of alleles in custom VCF files.
- API perldoc documentation added.

New in version 88 ***(March 2017)***

- ensembl-vep is now the officially supported version of VEP
- Documentation updated to reflect switch to ensembl-vep. See the Ensembl archive site for documentation of the obsolete ensembl-tools VEP.
- The VEP script is now named simply **`vep`** (formerly **`variant_effect_predictor.pl`** or **`vep.pl`**)
- Directly use tabix-indexed GFF/GTF files as annotation sources
- Allele-specific reporting of frequencies (--af and more) and custom VCF annotations
- --check_existing now compares alleles by default, disable with --no_check_alleles
- Report the highest allele frequency observed in any population from 1000 genomes, ESP or ExAC using --max_af
- Get genomic HGVS nomenclature with --hgvsg
- Find the gene or transcript with the nearest transcription start site (TSS) to each input variant with --nearest
- filter_vep supports field/field comparisons e.g. AFR_AF > #EUR_AF
- Exclude predicted (XM and XR) transcripts when using RefSeq or merged cache with --exclude_predicted
- Filter transcripts used for annotation with --transcript_filter
- pileup input format no longer supported

Older versions (ensembl-tools) - until version 87:

Versions of VEP up to and including 87 were released as part of the ensembl-tools package. See download links above.

New in version 87 ***(December 2016)***

- Shiny new code available for beta testing!
- Some minor speed optimisations
- Improve checks for valid chromosome names in input
- Haplosaurus beta released - generate whole-transcript haplotype sequences from phased genotype data

New in version 86 ***(October 2016)***

- Chromosome synonyms supported when using VEP caches; may be loaded manually with --synonyms

New in version 85 ***(July 2016)***

- --pick now uses translated length instead of genomic transcript length
- Support for epigenomes in regulatory features

New in version 84 ***(March 2016)***

- Add tab-delimited output option
- Add transcript flags indicating if the transcript is 5'- or 3'-incomplete
- Improve annotation of long variants where invariant parts of the alternate allele overlap splice regions

New in version 83 ***(December 2015)***

- Speed:
  - Basic consequence calculations up to 2x faster than version 82
  - HGVS calculations up to 10x faster
  - FASTA sequence retrieval implements caching
- Add ExAC project frequencies with --af_exac
- APPRIS isoform annotations now available with --appris and used by --pick and others to prioritise VEP annotations

New in version 82 ***(September 2015)***

- Faster FASTA file access using Bio::DB::HTS/htslib and bgzipped FASTA files
- Flag genes with phenotype associations
- Some plugins now available for use via the web and REST interfaces

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

New in version 80 ***(May 2015)***

- Flag added indicating if an overlapping known variant is associated with a phenotype, disease or trait
- HGVS notations are now 3'-shifted by default (use --shift_hgvs to force enable/disable)
- Source version information added to caches; see output file headers or use --show_cache_info
- Get the variant class using --variant_class
- CCDS status added to categories used by --pick flag (and others)

New in version 79 ***(March 2015)***

- Focus on performance and stability: ~100% faster than version 78 and a new test suite
- New guide to getting VEP running faster
- 1000 Genomes Phase 3 data available in GRCh37 cache download (GRCh38 coming soon, see docs to access now)
- VCF output has changed slightly to match output from other tools
- Impact modifier added for each consequence type

New in version 78 ***(December 2014)***

- Customise --pick using --pick_order
- Get transcript support level using --tsl

New in version 77 ***(October 2014)***

- Get the SO feature type of regulatory features using --regulatory and --biotype

New in version 76 ***(August 2014)***

- VEP now supports caches from multiple assemblies (--assembly) on the same software version - e.g. human builds GRCh37 and GRCh38
- Protein identifiers from UniProt (SWISSPROT, TrEMBL and UniParc) now available using --uniprot
- VEP can generate JSON output using --json
- Two new analysis set options - --gencode_basic and the merged Ensembl/RefSeq cache (--merged)
- Non-RefSeq transcripts now excluded by default when using the RefSeq or merged cache; use --all_refseq to include them
- Let VEP pick one consequence per variant allele using --pick_allele
- Allele now included alongside frequency for 1000 Genomes (--af_1kg) and ESP (--af_esp) data
- Not strictly script-related, but the VEP REST API has come out of beta!

New in version 75 ***(February 2014)***

- let VEP pick one consequence per variant for you using --pick; includes all transcript-specific data
- gene symbol available in RefSeq cache and when using --refseq
- Installation and use of RefSeq cache improved - remember to use --refseq with your RefSeq cache!
- Added --cache_version option, primarily to aid Ensembl Genomes users.

New in version 74 ***(December 2013)***

- retrieve the humDiv PolyPhen prediction instead of humVar using --humdiv
- source for gene symbol available with --symbol

New in version 73 ***(August 2013)***

- NHLBI-ESP frequencies available in cache (--af_esp)
- Pubmed IDs for cited existing variants available in cache (--pubmed)
- Convert your cache to use tabix - much faster when retrieving co-located existing variants!
- The installer can now update the VEP to the latest version and install FASTA files
- **--hgnc** replaced by --symbol for non-human compatibility
- HGVS strings are now part URI-escaped to avoid "=" sign clashes
- use --allele_number to identify input alleles by their order in the VCF ALT field
- use --total_length to give the total length of cDNA, CDS and protein sequences
- add data from VCF INFO fields when using custom annotations

New in version 72 ***(June 2013)***

- Speed and stability improvements when using forking
- Filter VEP results using filter_vep.pl

New in version 71 ***(April 2013)***

- SIFT predictions now available for Chicken, Cow, Dog, Human, Mouse, Pig, Rat and Zebrafish
- View summary statistics for VEP runs in [output]_summary.html
- Generate HTML output using --html
- Support for simple tab-delimited format for input of structural variant data
- Cache now contains clinical significance statuses from dbSNP for human variants

- **NOTE:** VEP version numbers have now (from release 71) changed to match Ensembl release numbers.

New in version 2.8 ***(December 2012)***

- Easily filter out common human variants with --filter_common
- 1000 Genomes continental population frequencies now stored in cache files

New in version 2.7 ***(October 2012)***

- build VEP cache files offline from GTF and FASTA files
- support for using FASTA files for sequence lookup in HGVS notations in offline/cache modes

New in version 2.6 ***(July 2012)***

- support for structural variant consequences
- Sequence Ontology (SO) consequence terms now default
- script runtime 3-4x faster when using forking
- 1000 Genomes global MAF available in cache files
- improved memory usage

New in version 2.5 ***(May 2012)***

- SIFT and PolyPhen predictions now available for RefSeq transcripts
- retrieve cell type-specific regulatory consequences
- consequences can be retrieved based on a single individual's genotype in a VCF input file
- find overlapping structural variants
- Condel support removed from main script and moved to a plugin

New in version 2.4 ***(February 2012)***

- offline mode and new installer script make it easy to use the VEP without the usual dependencies
- output columns configurable using the --fields flag
- VCF output support expanded, can now carry all fields
- output affected exon and intron numbers with --numbers
- output overlapping protein domains using --domains
- enhanced support for LRGs
- plugins now work on variants called as intergenic

New in version 2.3 ***(December 2011)***

- add custom annotations from tabix-indexed files (BED, GFF, GTF, VCF, bigWig)
- add new functionality to the VEP with user-written plugins
- filter input on consequence type

New in version 2.2 ***(September 2011)***

- SIFT, PolyPhen and Condel predictions and regulatory features now accessible from the cache
- support for calling consequences against RefSeq transcripts
- variant identifiers (e.g. dbSNP rsIDs) and HGVS notations supported as input format
- variants can now be filtered by frequency in HapMap and 1000 genomes populations
- script can be used to convert files between formats (Ensembl/VCF/Pileup/HGVS to Ensembl/VCF/Pileup)
- large amount of code moved to API modules to ensure consistency between web and script VEP
- memory usage optimisations
- VEP script moved to ensembl-tools repo
- Added --canonical, --per_gene and --no_intergenic options

New in version 2.1 ***(June 2011)***

- ability to use local file cache in place of or alongside connecting to an Ensembl database
- significant improvements to speed of script
- whole-genome mode now default (no disadvantage for smaller datasets)
- improved status output with progress bars
- regulatory region consequences now reinstated and improved
- modification to output file - Transcript column is now Feature, and is followed by a Feature_type column

New in version 2.0 ***(April 2011)***

- support for SIFT, PolyPhen and Condel missense predictions in human
- per-allele and compound consequence types
- support for Sequence Ontology (SO) and NCBI consequence terms
- modified output format
  - support for new output fields in Extra column
  - header section contains information on database and software versions
  - codon change shown in output
  - CDS position shown in output
  - option to output Ensembl protein identifiers
  - option to output HGVS nomenclature for variants
- support for gzipped input files
- enhanced configuration options, including the ability to read configuration from a file

- verbose output now much more useful
- whole-genome mode now more stable
- finding existing co-located variations now ~5x faster

Requirements

VEP requires:

- **gcc**, **g++** and **make**
- **Perl** version **5.10** or above recommended (tested on 5.10, 5.14, 5.18, 5.22, 5.26)
- **Perl** packages:
  - Archive::Zip
  - DBD::mysql
  - DBI

  See this guide for more information on how to install perl modules.
  Additional libraries can be installed for extra features and enhancements but they are not required to run VEP in most of the use cases.

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

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

Installation

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

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

```
perl INSTALL.pl
```

Additional non-essential components and enhancements must be installed manually.

Software components installed

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

If you already have the latest version of the API installed you do not need to run the installer, although it can be used to simply update your API version (with post-release patches applied), and retrieve cache and FASTA files. The installer downloads the API within the VEP directory and will not affect any other Ensembl API installations.

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

Running the installer

The installer is run on the command line as follows:

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

Follow on-screen prompts and note warnings of any files which will be deleted/overwritten

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

Flag	Alternate	Description
`--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.
`--AUTO`	`-a`	Run installer without prompts. Use the following options to specify parts to install: • **a** (API + Bio::DB::HTS/htslib) • **l** (Bio::DB::HTS/htslib only) • **c** (cache) • **f** (FASTA) • **p** (plugins) — Require the use of the --PLUGINS flag to list the plugin(s) to install. e.g. for API and cache: `perl INSTALL.pl --AUTO ac`
`--CACHE_VERSION [version]`		By default the installer will download the latest version of VEP caches and FASTA files (currently 103). You can force the script to install a different version, but there is no guarantee that a version of the API will be compatible with a different version of the cache.
`--CACHEDIR [dir]`	`-c`	By default the script will install the cache files in the ".vep" subdirectory in your home area. This option configures where cache files are installed. The --dir_cache flag must be passed when running the VEP if a non-default cache directory is given:

		`./vep --dir_cache [dir]`
--DESTDIR [dir]	-d	By default the script will install the API modules in a subdirectory of the current directory named "Bio". Using this option you can configure where the Bio directory is created. If something other than the default is used, this directory must either be added to your PERL5LIB environment variable when running the VEP, or included using perl's -I flag: `perl -I [dir] vep`
--NO_HTSLIB	-l	Don't attempt to install Bio::DB::HTS/htslib
--NO_TEST		Don't run API tests - useful if you know a harmless failure will prevent continuation of the installer
--NO_UPDATE	-n	By default the script will check for new versions or updates of the VEP. Using this option will skip this check.

--PLUGINS	-g	Comma-separated list of plugins to install when using `--AUTO`. To install all available plugins, use `--PLUGINS all`.
		`# List the available plugins:`
		`perl INSTALL.pl -a p --PLUGINS list`
		`# Download/install all the available plugins:`
		`perl INSTALL.pl -a p --PLUGINS all`
		`# Download/install a defined list of plugins, e.g.:`
		`perl INSTALL.pl -a p --PLUGINS dbNSFP,CADD,G2P`

--PLUGINSDIR [dir]	-r	By default the script will install the plugins files in the "Plugins" subdirectory of the `--CACHEDIR` directory. This option configures where the plugins files are installed. The --dir_plugins flag must be passed when running the VEP if a non-default plugins directory is given: `./vep --dir_plugins [dir]`
--PREFER_BIN	-p	Use this if the installer fails with out of memory errors.
--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!
--QUIET	-q	Don't write any status output when using `--AUTO`.

Additional components

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

Typically, you will use cpanm to install modules locally in your home directories; this shows how to set up a path for perl modules and install one there:

```
mkdir -p $HOME/cpanm
export PERL5LIB=$PERL5LIB:$HOME/cpanm/lib/perl5
cpanm -l $HOME/cpanm Set::IntervalTree
```

To make the change to `PERL5LIB` permanent, it is recommended to add the `export` line to your `$HOME/.bashrc` or `$HOME/.profile`.

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

Bio::DB::BigFile

In order for VEP to be able to access bigWig format custom annotation files, the Bio::DB::BigFile perl module is required. Installation involves downloading and compiling the kent source tree. The current version of the kent source tree does not work correctly with Bio::DB::BigFile, so it is necessary to install an archive version known to work (v335).

1. Download and unpack the kent source tree

```
wget https://github.com/ucscGenomeBrowser/kent/archive/v335_base.tar.gz
tar xzf v335_base.tar.gz
```

2. Set up some environment variables; these are required only temporarily for this installation process

```
export KENT_SRC=$PWD/kent-335_base/src
export MACHTYPE=$(uname -m)
export CFLAGS="-fPIC"
export MYSQLINC=`mysql_config --include | sed -e 's/^-I//g'`
export MYSQLLIBS=`mysql_config --libs`
```

3. Modify kent build parameters

```
cd $KENT_SRC/lib
echo 'CFLAGS="-fPIC"' > ../inc/localEnvironment.mk
```

4. Build kent source

```
make clean && make
cd ../jkOwnLib
make clean && make
```

If either of these steps fail, you may have some missing dependencies. Known common missing dependencies are libpng and libssl; these may be installed, for example, with `apt-get` on Ubuntu. If you do not have sudo access you may have to ask your sysadmin to install any missing dependencies.

```
sudo apt-get install libpng-dev libssl-dev
```

On Mac OSX you may use brew; the openssl libraries also need to be symbolically linked to a different path:

```
brew install libpng openssl
cd /usr/local/include
ln -s ../opt/openssl/include/openssl .
cd -
```

5. On some systems (e.g. Mac OSX), a compiled file is placed in a path that Bio::DB::BigFile cannot find. You can correct this with:

```
ln -s $KENT_SRC/lib/x86_64/* $KENT_SRC/lib/
```

6. We'll now use cpanm to install the perl module for Bio::DB::BigFile itself. See above for guidance on this. In this example we're going to install the module to a path within your home directory. In order to do this we must modify the paths that perl looks in to find modules by adding to the `PERL5LIB` environment module. To make this change permanent you must add the `export` line to your `$HOME/.bashrc` or `$HOME/.profile`.

```
mkdir -p $HOME/cpanm
export PERL5LIB=$PERL5LIB:$HOME/cpanm/lib/perl5
cpanm -l $HOME/cpanm Bio::DB::BigFile
```

If you are prompted for the path to the kent source tree, that means something didn't go right in the compilation above. Double check that `$KENT_SRC/lib/jkweb.a` exists and is not found instead at e.g. `$KENT_SRC/lib/x86_64/jkweb.a`. You may copy or link the file (and the other files in that directory) to the former path.

```
ln -s $KENT_SRC/lib/x86_64/* $KENT_SRC/lib/
```

7. You should now be able to successfully run the appropriate test in the VEP package:

```
perl -Imodules t/AnnotationSource_File_BigWig.t
```

Using VEP in Mac OS

Installing VEP on Mac OS is slightly trickier than other Linux-based systems, and will require additional dependancies.
These instructions will guide you through the setup of **Perlbrew**, **Homebrew**, **MySQL** and other dependancies that will allow for a clean installation of VEP on your Mac OS system.

These instructions have been tested on **macOS High Sierra (10.13)** and **macOS Sierra (10.12)**.
Older versions may require additional tweaks, however we shall endeavor to keep these instructions up to date for future versions of MacOS.

Prerequisite Setup

List of prerequisites: **XCode**, **GCC**, **Perlbrew**, **Cpanm**, **Homebrew**, **mysql**, **DBI**, **DBD::mysql**

XCode and GCC

VEP requires XCode and GCC for installation purposes. Fortunately, recent versions of macOS will look for (and attempt to install if required) both of these when you run the following command:

```
gcc -v
```

Perlbrew

We recommend using Perlbrew to install a new version of Perl on your mac, to prevent messing with the vendor perl too much. This can be done with the following command:

```
curl -L http://install.perlbrew.pl | bash

echo 'source $HOME/perl5/perlbrew/etc/bashrc' >> ~/.bash_profile
```

At this point, PLEASE RESTART YOUR TERMINAL WINDOW to allow for the perlbrew changes to take effect.

We recommend installing Perl version **5.26.2** to run VEP, and installing cpanm to handle the installation of perl modules.
These steps can be completed with the commands:

```
perlbrew install -j 5 --as 5.26.2 --thread --64all -Duseshrplib perl-5.26.2 --notest
perlbrew switch 5.26.2
perlbrew install-cpanm
```

Homebrew

This package management system for Mac OS would make the installation of the next prerequisite (i.e. xs) easier.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

xz

VEP requires the installation of xz, a data-compression utility. The easiest way to install the xz package is through homebrew:

```
brew install xz
```

MySQL

In order to connect to the Ensembl databases, a collection of MySQL related dependancies are required. Fortunately, these can be installed neatly with **Homebrew** and **Cpanm**:

```
brew install mysql
cpanm DBI
cpanm DBD::mysql
```

Installing BioPerl

On some versions of macOS, the VEP installer fails to cleanly install BioPerl, so a manual install will prevent issues:

```
curl -O https://cpan.metacpan.org/authors/id/C/CJ/CJFIELDS/BioPerl-1.6.924.tar.gz
tar zxvf BioPerl-1.6.924.tar.gz
echo 'export PERL5LIB=${PERL5LIB}:##PATH_TO##/bioperl-1.6.924' >> ~/.bash_profile
```

where *##PATH_TO##/bioperl-1.6.924* refers to the location of the newly unzipped BioPerl directory.

Final Dependancies

Installing the following Perl modules with cpanm will allow for full VEP functionality:

```
cpanm Test::Differences Test::Exception Test::Perl::Critic Archive::Zip PadWalker Error Devel::Cycle Role::Tiny::With Module::Build

export DYLD_LIBRARY_PATH=/usr/local/mysql/lib/:$DYLD_LIBRARY_PATH
```

Installing VEP

And that should be that! You should now be able to install VEP using the installer:

```
git clone https://github.com/ensembl/ensembl-vep
cd ensembl-vep
perl INSTALL.pl --NO_TEST
```

Using VEP in Windows

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

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

Virtual machines

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

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

Perl

If Perl is installed on Windows, VEP can be setup. However this may require installation of dependent modules. We recommend using Docker to run VEP on Windows.

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

   ```
   cd Downloads/ensembl-vep-release-103
   ```

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

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

Docker

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

The VEP Docker image uses ubuntu:18.04 as base image.

Commands to download the VEP Docker image (need to download and install the docker client beforehand):

```
docker pull ensemblorg/ensembl-vep
docker run -t -i ensemblorg/ensembl-vep ./vep
```

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

The following is a brief example showing how to use a directory on your local (host) machine to store cache data for VEP.

```
# Create a directory on your machine:
mkdir $HOME/vep_data
```

```
# Make sure that the created directory on your machine has read and write access granted
# so the docker container can write in the directory (VEP output):
chmod a+rwx $HOME/vep_data

docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep
```

Cache and Plugins installation

You will now be prompted by the installer if you wish to re-install the API. Type "n" followed by enter to skip to cache installation. You will be presented with a list of species; type the number for your species/assembly of interest and press enter. Your data will now download and unpack; this may take a while.

If you wish to retrieve HGVS annotations it is recommended to also download the FASTA file for your species. To do this, at the next prompt type "0" and press enter. You may skip the plugin installation also.

The above process may also be performed in one command; for example, to set up the cache and corresponding FASTA for human GRCh38:

```
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep perl INSTALL.pl -a cf -s homo_sapiens -y GRCh38
```

If you wish to include the VEP plugins, add the '**p**' value to the **-a** flag and the --PLUGINS (or **-g**) flag as well:

```
# Install all the available plugins:
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep perl INSTALL.pl -a cfp -s homo_sapiens -y GRCh38 -g all
# or install a defined list of plugins:
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep perl INSTALL.pl -a cfp -s homo_sapiens -y GRCh38 -g dbNSFP,CADD,G2P
```

The installer has now downloaded this data to **$HOME/vep_data** (and **$HOME/vep_data/Plugins** for the VEP plugins). VEP will automatically detect caches downloaded in this folder as it is mapped to VEP's default directory within the Docker instance.

```
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep
./vep -i examples/homo_sapiens_GRCh38.vcf --cache
```

Mounted volume - recommended data structure

i.e. VEP data structure outside the Docker container

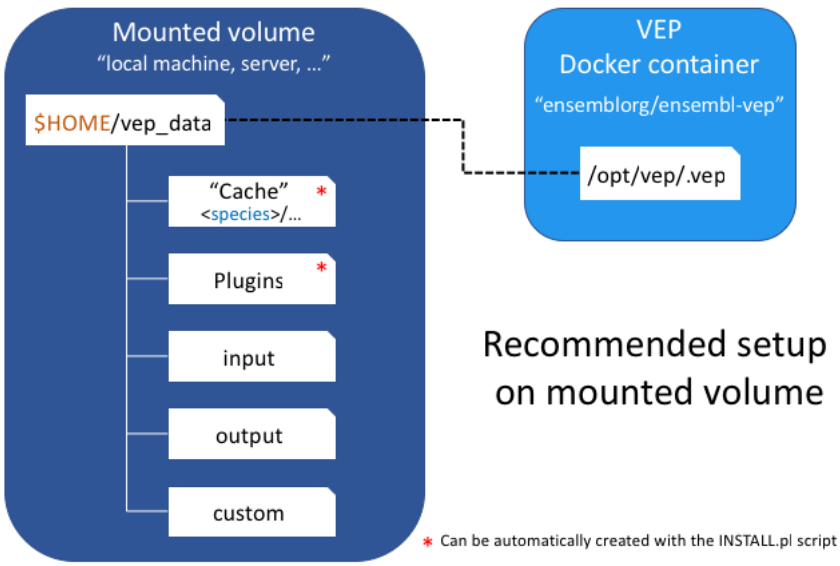


Diagram representing a recommended data file structure for the mounted volume

Here is an example of how you can run VEP using the setup presented in the image above (providing that the cache, the dbNSFP plugin and its data file have been downloaded):

```
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep

# Example of VEP command line:
./vep --cache --offline --format vcf --vcf --force_overwrite \
--dir_cache /opt/vep/.vep/ \
--dir_plugins /opt/vep/.vep/Plugins/ \
--input_file /opt/vep/.vep/input/my_input.vcf \
--output_file /opt/vep/.vep/output/my_output.vcf \
--custom /opt/vep/.vep/custom/my_extra_data.bed,BED_DATA,bed,exact,1 \
--plugin dbNSFP,/opt/vep/.vep/Plugins/dbNSFP.gz,ALL
```

Update from a previous version

1. Update your docker container

```
# List containers
docker ps -a
# e.g.
```

```
CONTAINER ID     IMAGE                          COMMAND          CREATED              STATUS                          PORTS          NAMES
d64055ffe9e9     ensemblorg/ensembl-vep         "/bin/bash"      About a minute ago   Exited (0) 59 seconds ago                      tender

# Stop and remove old container
docker stop tender_ritchie
docker rm tender_ritchie

# Update the container
docker pull ensemblorg/ensembl-vep
```

2. Update your cache

```
# Install the new cache through the VEP INSTALL.pl script (see "Cache installation" section above)
docker run -t -i -v $HOME/vep_data:/opt/vep/.vep ensemblorg/ensembl-vep perl INSTALL.pl -a c

# Or you can install the cache manually
cd $HOME/vep_data
curl -O /release-103/variation/vep/homo_sapiens_vep_103_GRCh38.tar.gz
tar xzf homo_sapiens_vep_103_GRCh38.tar.gz
```

Input

Both the web and script version of VEP can use the same input formats. Formats can be auto-detected by the VEP script, but must be manually selected when using the web interface.

VEP can use different input formats:

Default VEP input

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

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

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

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

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

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

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

VCF

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

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

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

This problem can be overcome with the following:

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
3. using --minimal and --allele_number (see Complex VCF entries).

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

```
Ref: a t C g a // C is the reference base
1 : a t G g a // C base is a G in individual 1
2 : a t - g a // C base is deleted w.r.t. the reference in individual 2
3 : a t CAg a // A base is inserted w.r.t. the reference sequence in individual 3
```

Individual 1

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

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

And in Ensembl format:

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

Individual 2

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

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

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

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

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

```
20_3_C/-
```

Individual 3

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

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

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

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

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

```
20_3_-/A
```

Using VCF format output, or adding unique identifiers to the input (in the third VCF column), can mitigate this issue.

Complex VCF entries

For VCF entries with multiple alternate alleles, VEP will only trim the leading base from alleles if **all** REF and ALT alleles start with the same base:

```
20   3   .   C   CAAG,CAAGAAG   .   PASS   .
```

This will be considered internally by VEP as equivalent to:

```
20   4   3   -/AAG/AAGAAG   +
```

Now consider the case where a single VCF line contains a representation of both a SNV and an insertion:

```
20   3   .   C   CAAAG,G   .   PASS   .
```

Here the input alleles will remain unchanged, and VEP will consider the first REF/ALT pair as a substitution of C for CAAG, and the second as a C/G SNV:

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

To modify this behaviour, VEP script users may use --minimal. This flag forces VEP to consider each REF/ALT pair independently, trimming identical leading and trailing bases from each as appropriate. Since this can lead to confusing output regarding coordinates etc, it is not the default behaviour. It is recommended to use the --allele_number flag to track the correspondence between alleles as input and how they appear in the output.

VCF - Structural variants

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

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

Examples of structural variants encoded in tab-delimited format:

```
1    160283    471362    DUP   +  sv1
1    1385015   1387562   DEL   +  sv2
```

Examples of structural variants encoded in VCF format:

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

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

HGVS identifiers

See https://varnomen.hgvs.org for details. These must be relative to genomic or Ensembl transcript coordinates.

It also is possible to use RefSeq transcripts in both the web interface and the VEP script (see script documentation): this works for RefSeq transcripts that align to the genome correctly.

Examples:

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

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

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

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

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

HGVS notations may also be given in LRG coordinates:

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

Variant identifiers

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

Genomic SPDI notation

VEP can also support genomic SPDI notation which uses four fields delimited by colons S:P:D:I (Sequence:Position:Deletion:Insertion). See here for details.

Examples:

```
NC_000016.10:68684738:G:A
NC_000017.11:43092199:GCTTTT:
NC_000013.11:32315789::C
NC_000016.10:68644746:AA:GTA
16:68684738:2:AC
```

REST-style regions

VEP's region REST endoint requires variants are described as `[chr]:[start]-[end]:[strand]/[allele]`. This follows the same conventions as the default input format described above, with the key difference being that this format does not require the reference (REF) allele to be included; VEP will look up the reference allele using either a provided FASTA file (preferred) or Ensembl core database. Strand is optional and defaults to 1 (forward strand).

```
# SNP
5:140532-140532:1/C

# SNP (reverse strand)
14:19584687-19584687:-1/T

# insertion
1:881907-881906:1/C

# 5bp deletion
2:946507-946511:1/-
```

Output

VEP can return the results in different formats:

- Default VEP output
- Tab-delimited output
- VCF
- JSON output

Along with the results VEP computes and returns some statistics.

Default VEP output

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

1. **Uploaded variation** - as chromosome_start_alleles
2. **Location** - in standard coordinate format (chr:start or chr:start-end)
3. **Allele** - the variant allele used to calculate the consequence
4. **Gene** - Ensembl stable ID of affected gene
5. **Feature** - Ensembl stable ID of feature
6. **Feature type** - type of feature. Currently one of Transcript, RegulatoryFeature, MotifFeature.
7. **Consequence** - consequence type of this variant
8. **Position in cDNA** - relative position of base pair in cDNA sequence
9. **Position in CDS** - relative position of base pair in coding sequence
10. **Position in protein** - relative position of amino acid in protein
11. **Amino acid change** - only given if the variant affects the protein-coding sequence
12. **Codon change** - the alternative codons with the variant base in upper case
13. **Co-located variation** - known identifier of existing variant
14. **Extra** - this column contains extra information as key=value pairs separated by ";", see below.

Other output fields:

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

- **gnomAD_AMR_AF** - Frequency of existing variant in gnomAD exomes American population
- **gnomAD_ASJ_AF** - Frequency of existing variant in gnomAD exomes Ashkenazi Jewish population
- **gnomAD_EAS_AF** - Frequency of existing variant in gnomAD exomes East Asian population
- **gnomAD_FIN_AF** - Frequency of existing variant in gnomAD exomes Finnish population
- **gnomAD_NFE_AF** - Frequency of existing variant in gnomAD exomes Non-Finnish European population
- **gnomAD_OTH_AF** - Frequency of existing variant in gnomAD exomes combined other combined populations
- **gnomAD_SAS_AF** - Frequency of existing variant in gnomAD exomes South Asian population
- **MAX_AF** - Maximum observed allele frequency in 1000 Genomes, ESP and gnomAD
- **MAX_AF_POPS** - Populations in which maximum allele frequency was observed
- **CLIN_SIG** - ClinVar clinical significance of the dbSNP variant
- **BIOTYPE** - Biotype of transcript or regulatory feature
- **APPRIS** - Annotates alternatively spliced transcripts as primary or alternate based on a range of computational methods. NB: not available for GRCh37
- **TSL** - Transcript support level. NB: not available for GRCh37
- **PUBMED** - Pubmed ID(s) of publications that cite existing variant
- **SOMATIC** - Somatic status of existing variant(s); multiple values correspond to multiple values in the Existing_variation field
- **PHENO** - Indicates if existing variant is associated with a phenotype, disease or trait; multiple values correspond to multiple values in the Existing_variation field
- **GENE_PHENO** - Indicates if overlapped gene is associated with a phenotype, disease or trait
- **ALLELE_NUM** - Allele number from input; 0 is reference, 1 is first alternate etc
- **MINIMISED** - Alleles in this variant have been converted to minimal representation before consequence calculation
- **PICK** - indicates if this block of consequence data was picked by --flag_pick or --flag_pick_allele
- **BAM_EDIT** - Indicates success or failure of edit using BAM file
- **GIVEN_REF** - Reference allele from input
- **USED_REF** - Reference allele as used to get consequences
- **REFSEQ_MATCH** - the RefSeq transcript match status; contains a number of flags indicating whether this RefSeq transcript matches the underlying reference sequence and/or an Ensembl transcript (more information).
  - *rseq_3p_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the 3' UTR of the RefSeq model with respect to the primary genome assembly (e.g. GRCh37/GRCh38).
  - *rseq_5p_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the 5' UTR of the RefSeq model with respect to the primary genome assembly.
  - *rseq_cds_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. Specifically, there is a mismatch in the CDS of the RefSeq model with respect to the primary genome assembly.
  - *rseq_ens_match_cds:* signifies that for the RefSeq transcript there is an overlapping Ensembl model that is identical across the CDS region only. A CDS match is defined as follows: the CDS and peptide sequences are identical and the genomic coordinates of every translatable exon match. Useful related attributes are: rseq_ens_match_wt and rseq_ens_no_match.
  - *rseq_ens_match_wt:* signifies that for the RefSeq transcript there is an overlapping Ensembl model that is identical across the whole transcript. A whole transcript match is defined as follows: 1) In the case that both models are coding, the transcript, CDS and peptide sequences are all identical and the genomic coordinates of every exon match. 2) In the case that both transcripts are non-coding the transcript sequences and the genomic coordinates of every exon are identical. No comparison is made between a coding and a non-coding transcript. Useful related attributes are: rseq_ens_match_cds and rseq_ens_no_match.
  - *rseq_ens_no_match:* signifies that for the RefSeq transcript there is no overlapping Ensembl model that is identical across either the whole transcript or the CDS. This is caused by differences between the transcript, CDS or peptide sequences or between the exon genomic coordinates. Useful related attributes are: rseq_ens_match_wt and rseq_ens_match_cds.
  - *rseq_mrna_match:* signifies an exact match between the RefSeq transcript and the underlying primary genome assembly sequence (based on a match between the transcript stable id and an accession in the RefSeq mRNA file). An exact match occurs when the underlying genomic sequence of the model can be perfectly aligned to the mRNA sequence post polyA clipping.
  - *rseq_mrna_nonmatch:* signifies a non-match between the RefSeq transcript and the underlying primary genome assembly sequence. A non-match is deemed to have occurred if the underlying genomic sequence does not have a perfect alignment to the mRNA sequence post polyA clipping. It can also signify that no comparison was possible as the model stable id may not have had a corresponding entry in the RefSeq mRNA file (sometimes happens when accessions are retired or changed). When a non-match occurs one or several of the following transcript attributes will also be present to provide more detail on the nature of the non-match: rseq_5p_mismatch, rseq_cds_mismatch, rseq_3p_mismatch, rseq_nctran_mismatch, rseq_no_comparison
  - *rseq_nctran_mismatch:* signifies a mismatch between the RefSeq transcript and the underlying primary genome assembly sequence. This is a comparison between the entire underlying genomic sequence of the RefSeq model to the mRNA in the case of RefSeq models that are non-coding.
  - *rseq_no_comparison:* signifies that no alignment was carried out between the underlying primary genome assembly sequence and a corresponding RefSeq mRNA. The reason for this is generally that no corresponding, unversioned accession was found in the RefSeq mRNA file for the transcript stable id. This sometimes happens when accessions are retired or replaced. A second possibility is that the sequences were too long and problematic to align (though this is rare).
- **OverlapBP** - Number of base pairs overlapping with the corresponding structural variation feature
- **OverlapPC** - Percentage of corresponding structural variation feature overlapped by the given input
- **CHECK_REF** - Reports variants where the input reference does not match the expected reference
- **AMBIGUITY** - IUPAC allele ambiguity code

Example of VEP default output format:

```
11_224088_C/A    11:224088   A  ENSG00000142082  ENST00000525319  Transcript         missense_variant           742  716  239  T/N  aCc/aAc
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 v103.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_103_38 on ensembldb.ensembl.org
## Using cache in /homes/user/.vep/homo_sapiens/103_GRCh38
## Using API version 103, DB version 103
## polyphen version 2.2.2
## sift version sift5.2.2
## COSMIC version 78
```

```
## ESP version 20141103
## gencode version GENCODE 25
## genebuild version 2014-07
## HGMD-PUBLIC version 20162
## regbuild version 16
## assembly version GRCh38.p7
## ClinVar version 201610
## dbSNP version 147
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Feature_type : Type of feature - Transcript, RegulatoryFeature or MotifFeature
## Consequence : Consequence type
## cDNA_position : Relative position of base pair in cDNA sequence
## CDS_position : Relative position of base pair in coding sequence
## Protein_position : Relative position of amino acid in protein
## Amino_acids : Reference and variant amino acids
## Codons : Reference and variant codon sequence
## Existing_variation : Identifier(s) of co-located known variants
## Extra column keys:
## IMPACT : Subjective impact classification of consequence type
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
## FLAGS : Transcript quality flags
```

Tab-delimited output

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

Example of VEP tab-delimited output format:

```
#Uploaded_variation  Location   Allele  Gene             Feature          Feature_type  Consequence                          cDNA_positio
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000525319  Transcript    missense_variant                     742
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000534381  Transcript    downstream_gene_variant              -
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000529055  Transcript    downstream_gene_variant              -
11_224585_G/A        11:224585  A       ENSG00000142082  ENST00000529937  Transcript    intron_variant,NMD_transcript_variant  -
```

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

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --tab --fields "Uploaded_variation,Location,Allele,Gene"
```

VCF output

The VEP script can also generate VCF output using the --vcf flag.

Main information about the specificity of the VEP VCF output format:

- Consequences are added in the INFO field of the VCF file, using the key "**CSQ**" (you can change it using --vcf_info_field).
- Data fields are encoded separated by the character "**|**" (pipe). The order of fields is written in the VCF header. Unpopulated fields are represented by an empty string.
- Output fields in the "CSQ" INFO field can be configured by using --fields.
- Each prediction, for a given variant, is separated by the character "**,**" in the CSQ INFO field (e.g. when a variant overlaps more than 1 transcript)

Here is a list of the (default) fields you can find within the CSQ field:

```
Allele|Consequence|IMPACT|SYMBOL|Gene|Feature_type|Feature|BIOTYPE|EXON|INTRON|HGVSc|HGVSp|cDNA_position|CDS_position|Protein_position|Amino
```

Example of VEP command using the --vcf and --fields options:

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache --force_overwrite --vcf --fields "Allele,Consequence,Feature_type,Feature"
```

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

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

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

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

```
##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from Ensembl VEP. Format: Allele|Consequence|IMPACT|SYMBOL|Gene|Fe
#CHROM  POS       ID          REF  ALT  QUAL  FILTER  INFO
21      26978790  rs75377686  T    C    .     .       CSQ=C|missense_variant|MODERATE|MRPL39|ENSG00000154719|Transcript|ENST00000419219|prot
```

JSON output

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

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

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


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


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

Key	JSON equivalent(s)	Notes
Consequence	consequence_terms	
Gene	gene_id	

Feature	transcript_id, regulatory_feature_id, motif_feature_id	Consequences are grouped under the feature type they affect
ALLELE	variant_allele	
SYMBOL	gene_symbol	
SYMBOL_SOURCE	gene_symbol_source	
ENSP	protein_id	
OverlapBP	bp_overlap	
OverlapPC	percentage_overlap	
Uploaded_variation	id	
Location	seq_region_name, 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

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

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

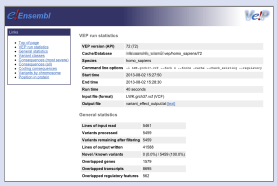
The page contains several sections:

General statistics

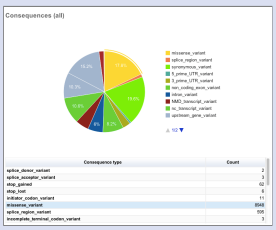
This section contains two tables. The first describes the cache and/or database used, the version of VEP, species, command line parameters, input/output files and run time. The second table contains information about the number of variants, and the number of genes, transcripts and regulatory features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

Charts and tables

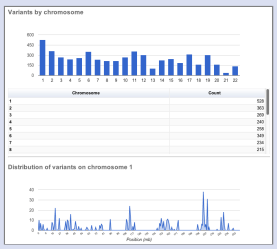
There then follows several charts, most with accompanying tables. Tables and charts are interactive; clicking on a row to highlight it in the table will highlight the relevant segment in the chart, and vice versa.



General statistics



Summary of called consequence types



Distribution of variants across chromosomes

Variant Effect Predictor ⚙ Running VEP

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

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

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

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

For optimum performance, download a cache file for your species of interest, using either the installer or by following the VEP Cache documentation, and run VEP with either the --cache or --offline option.

It is possible to run VEP connecting to the public Ensembl database servers in place of a cache. This can be adequate when annotating small files, but the database servers can become busy and slow. To enable this option, use --database

To run VEP with default options, use the following command:

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

where input.txt contains data in one of the compatible input formats and output.txt is the output file to be created.

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

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

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

Basic options

Flag	Alternate	Description	Incompatible with
--help		Display help message and quit	
--quiet	-q	Suppress warning messages.*Not used by default*	--verbose
--verbose	-v	Print out a bit more information while running. *Not used by default*	--quiet

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

--everything	-e	Shortcut flag to switch on all of the following: --sift b, --polyphen b, --ccds, --uniprot, --hgvs, --symbol, --numbers, --domains, --regulatory, --canonical, --protein, --biotype, --uniprot, --tsl, --appris, --gene_phenotype --af, --af_1kg, --af_esp, --af_gnomad, --max_af, --pubmed, --var_synonyms, --variant_class, --mane	
--species [species]		Species for your data. This can be the latin name e.g. "homo_sapiens" or any Ensembl alias e.g. "mouse". Specifying the latin name can speed up initial database connection as the registry does not have to load all available database aliases on the server. *Default = "homo_sapiens"*	
--assembly [name]	-a	Select the assembly version to use if more than one available. If using the cache, you must have the appropriate assembly's cache file installed. If not specified and you have only 1 assembly version installed, this will be chosen by default. *Default = use found assembly version*	
--input_file [filename]	-i	Input file name. If not specified, VEP will attempt to read from STDIN. Can use compressed file (gzipped).	
--input_data [string]	--id	Raw input data as a string. May be used, for example, to input a single rsID or HGVS notation quickly to vep: `--input_data rs699`	
--format [format]		Input file format - one of "ensembl", "vcf", "hgvs", "id", "region", "spdi". By default, VEP auto-detects the input file format. Using this option you can specify the input file is Ensembl, VCF, IDs, HGVS, SPDI or region format. Can use compressed version (gzipped) of any file format listed above. *Auto-detects format by default*	
--output_file [filename]	-o	Output file name. Results can write to STDOUT by specifying 'STDOUT' as the output file name - this will force quiet mode. *Default = "variant_effect_output.txt"*	
--force_overwrite	--force	By default, VEP will fail with an error if the output file already exists. You can force the overwrite of the existing file by using this flag. *Not used by default*	
--stats_file [filename]	--sf	Summary stats file name. This is an HTML file containing a summary of the VEP run - the file name must end ".htm" or ".html". *Default = "variant_effect_output.txt_summary.html"*	
--no_stats		Don't generate a stats file. Provides marginal gains in run time.	

`--stats_text`	Generate a plain text stats file in place of the HTML.
`--warning_file [filename]`	File name to write warnings and errors to. *Default = STDERR (standard error)*
`--max_sv_size`	Extend the maximum Structural Variant size VEP can process.
`--no_check_variants_order`	Permit the use of unsorted input files. However running VEP on unsorted input files slows down the tool and requires more memory.
`--fork [num_forks]`	Enable forking, using the specified number of forks. Forking can dramatically improve runtime. *Not used by default*

Cache options

Flag	Alternate	Description	Output fields	Incompatible with
`--cache`		Enables use of the cache. Add --refseq or --merged to use the refseq or merged cache, (if installed).		--database
`--dir [directory]`		Specify the base cache/plugin directory to use. *Default = "$HOME/.vep/"*		
`--dir_cache [directory]`		Specify the cache directory to use. *Default = "$HOME/.vep/"*		
`--dir_plugins [directory]`		Specify the plugin directory to use. *Default = "$HOME/.vep/"*		
`--offline`		Enable offline mode. No database connections will be made, and a cache file or GFF/GTF file is required for annotation. Add --refseq to use the refseq cache (if installed). *Not used by default*		--database --check_svs
`--fasta [file|dir]`	`--fa`	Specify a FASTA file or a directory containing FASTA files to use to look up reference sequence. The first time you run VEP with this parameter an index will be built which can take a few minutes. This is required if fetching HGVS annotations (--hgvs) or checking reference sequences (--check_ref) in offline mode (--offline), and optional with some performance increase in cache mode (--cache). See documentation for more details. *Not used by default*		
`--refseq`		Specify this option if you have installed the RefSeq cache in order for VEP to pick up the alternate cache directory. This cache contains transcript objects corresponding to RefSeq transcripts. Consequence output will be given relative to these transcripts in place of the default Ensembl transcripts (see documentation)	REFSEQ_MATCH, BAM_EDIT	--gencode_basic --merged
`--merged`		Use the merged Ensembl and RefSeq cache. Consequences are flagged with the SOURCE of each transcript used.	REFSEQ_MATCH, BAM_EDIT, SOURCE	--refseq
`--cache_version`		Use a different cache version than the assumed default (the VEP version). This should be used with Ensembl Genomes caches since their version numbers do not match Ensembl versions. For example, the VEP/Ensembl version may be 88 and the Ensembl Genomes version 35. *Not used by default*		
`--show_cache_info`		Show source version information for selected cache and quit		
`--buffer_size [number]`		Sets the internal buffer size, corresponding to the number of variants that are read in to memory simultaneously. Set this lower to use less memory at the expense of longer run time, and higher to use more memory with a faster run time. *Default = 5000*		

Other annotation sources

Flag	Alternate	Description	Output fields
`--plugin [plugin name]`		Use named plugin. Plugin modules should be installed in the Plugins subdirectory of the VEP cache directory (defaults to $HOME/.vep/). Multiple plugins can be used by supplying the --plugin flag multiple times. See plugin documentation. *Not used by default*	Plugin-dependent
`--custom [filename]`		Add custom annotation to the output. Files must be tabix indexed or in the bigWig format. Multiple files can be specified by supplying the --custom flag multiple times. See here for full details. *Not used by default*	SOURCE, Custom file dependent
`--gff [filename]`		Use GFF transcript annotations in [filename] as an annotation source. Requires a FASTA file of genomic sequence.*Not used by default*	SOURCE
`--gtf [filename]`		Use GTF transcript annotations in [filename] as an annotation source. Requires a FASTA file of genomic sequence.*Not used by default*	SOURCE
`--bam [filename]`		**ADVANCED** Use BAM file of sequence alignments to correct transcript models not derived from reference genome sequence. Used to correct RefSeq transcript models. Enables --use_transcript_ref; add --use_given_ref to override this behaviour. *Not used by default*	BAM_EDIT
`--use_transcript_ref`		By default VEP uses the reference allele provided in the input file to calculate consequences for the provided alternate allele(s). Use this flag to force VEP to replace the provided reference allele with sequence derived from the overlapped transcript. This is especially relevant when using the RefSeq cache, see documentation for more details. The GIVEN_REF and USED_REF fields are set in the output to indicate any change. *Not used by default*	GIVEN_REF, USED_REF
`--use_given_ref`		Using --bam or a BAM-edited RefSeq cache by default enables --use_transcript_ref; add this flag to override this behaviour and use the provided reference allele from the input. *Not used by default*	
`--custom_multi_allelic`		By default, comma separated lists found within the INFO field of custom annotation VCFs are assumed to be allele specific. For example, a variant with allele_string A/G/C with associated custom annotation 'single,double,triple' will associate triple with C, double with G and single with A. This flag instructs VEP to return all annotations for all alleles. *Not used by default*	

Output format options

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

`--fields [list]`		Configure the output format using a comma separated list of fields.
		Can only be used with tab (--tab) or VCF format (--vcf) output.
		For the tab format output, the selected 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) if the appropriate output is available (e.g. use --show_ref_allele to access 'REF_ALLELE'). Output remains tab-delimited.
		For the VCF format output, the selected fields are those present within the "CSQ" INFO field.
		
		Example of command for the tab output:
		
		`--tab --fields "Uploaded_variation,Location,Allele,Gene"`
		
		Example of command for the VCF format output:
		
		`--vcf --fields "Allele,Consequence,Feature_type,Feature"`
		
		Not used by default		

`--minimal`		Convert alleles to their most minimal representation before consequence calculation i.e. sequence that is identical between each pair of reference and alternate alleles is trimmed off from both ends, with coordinates adjusted accordingly. Note this may lead to discrepancies between input coordinates and coordinates reported by VEP relative to transcript sequences; to avoid issues, use --allele_number and/or ensure that your input variants have unique identifiers. The MINIMISED flag is set in the VEP output where relevant. *Not used by default*	MINIMISED	--individual

Output options

Flag	Alternate	Description	Output fields	Incompatible with
`--variant_class`		Output the Sequence Ontology variant class. *Not used by default*	VARIANT_CLASS	
`--sift [p|s|b]`		Species limited SIFT predicts whether an amino acid substitution affects protein function based on sequence homology and the physical properties of amino acids. VEP can output the **p**rediction term, **s**core or **b**oth. *Not used by default*	SIFT	--most_severe --summary
`--polyphen [p|s|b]`		Human only PolyPhen is a tool which predicts possible impact of an amino acid substitution on the structure and function of a human protein using straightforward physical and comparative considerations. VEP can output the **p**rediction term, **s**core or **b**oth. VEP uses the humVar score by default - use --humdiv to retrieve the humDiv score. *Not used by default*	PolyPhen	--most_severe --summary
`--humdiv`		Human only Retrieve the humDiv PolyPhen prediction instead of the default humVar. *Not used by default*	PolyPhen	
`--nearest [transcript|gene|symbol]`		Retrieve the transcript or gene with the nearest protein-coding transcription start site (TSS) to each input variant. Use "transcript" to retrieve the transcript stable ID, "gene" to retrieve the gene stable ID, or "symbol" to retrieve the gene symbol. Note that the nearest TSS may not belong to a transcript that overlaps the input variant, and more than one may be reported in the	NEAREST	

		case where two are equidistant from the input coordinates. Currently only available when using a cache annotation source, and requires the Set::IntervalTree perl module. *Not used by default*	
`--distance [bp_distance(,downstream_distance_if_different)]`		Modify the distance up and/or downstream between a variant and a transcript for which VEP will assign the upstream_gene_variant or downstream_gene_variant consequences. Giving one distance will modify both up- and downstream distances; prodiving two separated by commas will set the up- (5') and down- (3') stream distances respectively. *Default: 5000*	
`--overlaps`		Report the proportion and length of a transcript overlapped by a structural variant in VCF format.	
`--gene_phenotype`		Indicates if the overlapped gene is associated with a phenotype, disease or trait. See list of phenotype sources. *Not used by default*	GENE_PHENO
`--regulatory`		Look for overlaps with regulatory regions. VEP can also report if a variant falls in a high information position within a transcription factor binding site. Output lines have a Feature type of RegulatoryFeature or MotifFeature. *Not used by default*	MOTIF_NAME, MOTIF_POS, HIGH_INF_POS, MOTIF_SCORE_CHANGE
`--cell_type`		Report only regulatory regions that are found in the given cell type(s). Can be a single cell type or a comma-separated list. The functional type in each cell type is reported under CELL_TYPE in the output. To retrieve a list of cell types, use --cell_type list. *Not used by default*	CELL_TYPE
`--individual [all|ind list]`		Consider only alternate alleles present in the genotypes of the specified individual(s). May be a single individual, a comma-separated list or "all" to assess all individuals separately. Individual variant combinations homozygous for the given reference allele will not be reported. Each individual and variant combination is given on a separate line of output. Only works with VCF files containing individual genotype data; individual IDs are taken from column headers. *Not used by default*	IND, ZYG
`--phased`		Force VCF genotypes to be interpreted as phased. For use with plugins that depend on phased data. *Not used by default*	
`--allele_number`		Identify allele number from VCF input, where 1 = first ALT allele, 2 = second ALT allele etc. Useful when using --minimal *Not used by default*	ALLELE_NUM
`--show_ref_allele`		Adds the reference allele in the output. Mainly useful for the VEP "default" and tab-delimited output formats. *Not used by default*	REF_ALLELE
`--total_length`		Give cDNA, CDS and protein positions as Position/Length. *Not used by default*	
`--numbers`		Adds affected exon and intron numbering to to output. Format is Number/Total. *Not used by default*	EXON, INTRON
`--no_escape`		Don't URI escape HGVS strings. *Default = escape*	
`--keep_csq`		Don't overwrite existing CSQ entry in VCF INFO field. *Overwrites by default*	
`--vcf_info_field [CSQ|ANN|(other)]`		Change the name of the INFO key that VEP write the consequences to in its VCF output. Use "ANN" for compatibility with other tools such as snpEff. *Default: CSQ*	
`--terms [SO|display|NCBI]`	`-t`	The type of consequence terms to output. The Ensembl terms are described here. The Sequence Ontology is a joint effort by genome annotation centres to standardise descriptions of biological sequences. *Default = "SO"*	
`--no_headers`		Don't write header lines in output files. *Default = add headers*	
`--shift_3prime [0|1]`		Right aligns all variants relative to their associated transcripts prior to consequence calculation. An example using this option can be found here. *Default = 0*	
`--shift_genomic [0|1]`		Right aligns all variants, including intergenic variants, before consequence calculation	

	--most_severe --summary
	--shift_hgvs
	--shift_hgvs

		and updates the *Location* field. An example using this option can be found here. *Default = 0*		
`--shift_length`		Reports the distance each variant has been shifted when used in conjuction with --shift_3prime		

Identifiers

Flag	Alternate	Description	Output fields	Incompatible with
`--hgvs`		Add HGVS nomenclature based on Ensembl stable identifiers to the output. Both coding and protein sequence names are added where appropriate. To generate HGVS identifiers when using --cache or --offline you must use a FASTA file and --fasta. HGVS notations given on Ensembl identifiers are versioned. *Not used by default*	HGVSc, HGVSp, HGVS_OFFSET	
`--hgvsg`		Add genomic HGVS nomenclature based on the input chromosome name. To generate HGVS identifiers when using --cache or --offline you must use a FASTA file and --fasta. *Not used by default*	HGVSg	
`--spdi`		Add genomic SPDI notation. To generate SPDI when using --cache or --offline you must use a FASTA file and --fasta. *Not used by default*	SPDI	
`--shift_hgvs [0|1]`		Enable or disable 3' shifting of HGVS notations. When enabled, this causes ambiguous insertions or deletions (typically in repetetive sequence tracts) to be "shifted" to their most 3' possible coordinates (relative to the transcript sequence and strand) before the HGVS notations are calculated; the flag HGVS_OFFSET is set to the number of bases by which the variant has shifted, relative to the input genomic coordinates. Disabling retains the original input coordinates of the variant. *Default: 1 (shift)*		
`--transcript_version`		Add version numbers to Ensembl transcript identifiers		
`--protein`		Add the Ensembl protein identifier to the output where appropriate. *Not used by default*	ENSP	--most_severe --summary
`--symbol`		Adds the gene symbol (e.g. HGNC) (where available) to the output. *Not used by default*	SYMBOL, SYMBOL_SOURCE, HGNC_ID	--most_severe --summary
`--ccds`		Adds the CCDS transcript identifer (where available) to the output. *Not used by default*	CCDS	--most_severe --summary
`--uniprot`		Adds best match accessions for translated protein products from three UniProt-related databases (SWISSPROT, TREMBL and UniParc) to the output. *Not used by default*	SWISSPROT, TREMBL, UNIPARC	--most_severe --summary
`--tsl`		Adds the transcript support level for this transcript to the output. *Not used by default*	TSL	--most_severe --summary
`--appris`		Adds the APPRIS isoform annotation for this transcript to the output. *Not used by default*	APPRIS	--most_severe --summary
`--canonical`		Adds a flag indicating if the transcript is the canonical transcript for the gene. *Not used by default*	CANONICAL	--most_severe --summary
`--mane`		Adds a flag indicating if the transcript is the MANE Select transcript for the gene. *Not used by default*	MANE	--most_severe --summary
`--biotype`		Adds the biotype of the transcript or regulatory feature. *Not used by default*	BIOTYPE	--most_severe --summary
`--domains`		Adds names of overlapping protein domains to output. *Not used by default*	DOMAINS	--most_severe --summary
`--xref_refseq`		Output aligned RefSeq mRNA identifier for transcript. *Not used by default*	RefSeq	--most_severe --summary
`--synonyms [file]`		Load a file of chromosome synonyms. File should be tab-delimited with the primary identifier in column 1 and the synonym in column 2. Synonyms allow different chromosome identifiers to be used in the input file and any annotation source (cache, database, GFF, custom file, FASTA file). *Not used by default*		

Co-located variants

Flag	Alternate	Description	Output fields	Incompatible with
`--check_existing`		Checks for the existence of known variants that are co-located with your input. By default the alleles are compared and variants on an allele-specific basis - to compare only coordinates, use --no_check_alleles. Some databases may contain variants with unknown (null) alleles and these are included by default; to exclude them use --exclude_null_alleles. See this page for more details. *Not used by default*	Existing_variation, CLIN_SIG, SOMATIC, PHENO	
`--check_svs`		Checks for the existence of structural variants that overlap your input. Currently requires database access. *Not used by default*	SV	--offline
`--clin_sig_allele [1|0]`		Return allele specific clinical significance. Setting this option to 0 will provide all known clinical	CLIN_SIG	

	significance values at the given locus. *Default: 1 (Provide allele-specific annotations)*		
`--exclude_null_alleles`	Do not include variants with unknown alleles when checking for co-located variants. Our human database contains variants from HGMD and COSMIC for which the alleles are not publically available; by default these are included when using --check_existing, use this flag to exclude them. *Not used by default*		
`--no_check_alleles`	When checking for existing variants, by default VEP only reports a co-located variant if none of the input alleles are novel. For example, if your input variant 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 input variant 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 input variant. Use this flag to disable this behaviour and compare using coordinates alone. *Not used by default*		
`--af`	Add the global allele frequency (AF) from 1000 Genomes Phase 3 data for any known co-located variant to the output. For this and all --af_* flags, the frequency reported is for the **input allele** only, not necessarily the non-reference or derived allele. *Not used by default*	AF	
`--max_af`	Report the highest allele frequency observed in any population from 1000 genomes, ESP or gnomAD. *Not used by default*	MAX_AF, MAX_AF_POPS	--database
`--af_1kg`	Add allele frequency from continental populations (AFR,AMR,EAS,EUR,SAS) of 1000 Genomes Phase 3 to the output. Must be used with --cache. *Not used by default*	AFR_AF, AMR_AF, EAS_AF, EUR_AF, SAS_AF	--database
`--af_esp`	Include allele frequency from NHLBI-ESP populations. Must be used with --cache. *Not used by default*	AA_AF, EA_AF	--database
`--af_gnomad`	Include allele frequency from Genome Aggregation Database (gnomAD) exome populations. Note only data from the gnomAD exomes are included; to retrieve data from the additional genomes data set, see this guide. Must be used with --cache *Not used by default*	gnomAD_AF, gnomAD_AFR_AF, gnomAD_AMR_AF, gnomAD_ASJ_AF, gnomAD_EAS_AF, gnomAD_FIN_AF, gnomAD_NFE_AF, gnomAD_OTH_AF, gnomAD_SAS_AF	--database
`--af_exac`	Include allele frequency from ExAC project populations. Must be used with --cache. *Not used by default*	ExAC_AF, ExAC_Adj_AF, ExAC_AFR_AF, ExAC_AMR_AF, ExAC_EAS_AF, ExAC_FIN_AF, ExAC_NFE_AF, ExAC_OTH_AF, ExAC_SAS_AF	--database
`--pubmed`	Report Pubmed IDs for publications that cite existing variant. Must be used with --cache. *Not used by default*	PUBMED	--database
`--var_synonyms`	Report known synonyms for colocated variants. Must be used with --cache. *Not used by default*	VAR_SYNONYMS	--database
`--failed [0|1]`	When checking for co-located variants, by default VEP will exclude variants that have been flagged as failed. Set this flag to include such variants. *Default: 0 (exclude)*		

Filtering and QC options

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

Flag	Alternate	Description	Output fields	Incompatible with
`--gencode_basic`		Limit your analysis to transcripts belonging to the GENCODE basic set. This set has fragmented or problematic transcripts removed. *Not used by default*		--refseq
`--exclude_predicted`		When using the RefSeq or merged cache, exclude predicted transcripts (i.e. those with identifiers beginning with "XM_" or "XR_").		
`--transcript_filter`		**ADVANCED** Filter transcripts according to any arbitrary set of rules. Uses similar notation to filter_vep. You may filter on any key defined in the root of the transcript object; most commonly this will be "stable_id": `--transcript_filter "stable_id match N[MR]_"`		
`--check_ref`		Force VEP to check the supplied reference allele against the sequence stored in the Ensembl Core database or supplied FASTA file. Lines that do not match are skipped. *Not used by default*		--lookup_ref
`--lookup_ref`		Force overwrite the supplied reference allele with the sequence stored in the Ensembl Core database or supplied FASTA file. *Not used by default*		--check_ref
`--dont_skip`		Don't skip input variants that fail validation, e.g. those that fall on unrecognised sequences. Combining --check_ref with --dont_skip will add a CHECK_REF output field when the given reference does not match the underlying reference sequence.	CHECK_REF	
`--allow_non_variant`		When using VCF format as input and output, by default VEP will skip non-variant lines of input (where the ALT allele is null). Enabling this option the lines will be printed in the VCF output with no consequence data added.		
`--chr [list]`		Select a subset of chromosomes to analyse from your file. Any data not on this chromosome in the input will be skipped. The list can be comma separated, with "-" characters representing an interval.		

	For example, to include chromosomes 1, 2, 3, 10 and X you could use --chr 1-3,10,X *Not used by default*		
--coding_only	Only return consequences that fall in the coding regions of transcripts. *Not used by default*		--most_severe --summary
--no_intergenic	Do not include intergenic consequences in the output. *Not used by default*		--most_severe --summary
--pick	Pick one line or block of consequence data per variant, including transcript-specific columns. Consequences are chosen according to the criteria described here, and the order the criteria are applied may be customised with --pick_order. This is the best method to use if you are interested only in one consequence per variant. *Not used by default*		--most_severe --summary
--pick_allele	Like --pick, but chooses one line or block of consequence data per variant allele. Will only differ in behaviour from **--pick** when the input variant has multiple alternate alleles. *Not used by default*		--most_severe --summary
--per_gene	Output only the most severe consequence per gene. The transcript selected is arbitrary if more than one has the same predicted consequence. Uses the same ranking system as --pick. *Not used by default*		
--pick_allele_gene	Like --pick_allele, but chooses one line or block of consequence data per variant allele **and** gene combination. *Not used by default*		
--flag_pick	As per --pick, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*	PICK	--most_severe --summary
--flag_pick_allele	As per --pick_allele, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*	PICK	--most_severe --summary
--flag_pick_allele_gene	As per --pick_allele_gene, but adds the PICK flag to the chosen block of consequence data and retains others. *Not used by default*	PICK	
--pick_order [c1,c2,...,cN]	Customise the order of criteria (and the list of criteria) applied when choosing a block of annotation data with one of the following options: --pick, --pick_allele, --per_gene, --pick_allele_gene, --flag_pick, --flag_pick_allele, --flag_pick_allele_gene. See this page for the default order. Valid criteria are: [*canonical appris tsl biotype ccds rank length mane*]. e.g.: `--pick --pick_order tsl,appris,rank`		
--most_severe	Output only the most severe consequence per variant. Transcript-specific columns will be left blank. Consequence ranks are given in this table. To include regulatory consequences, use the **--regulatory** option in combination with this flag. *Not used by default*		--appris --biotype --canonical --ccds --coding_only --domains --flag_pick --flag_pick_allele --no_intergenic --numbers --pick --pick_allele --polyphen --protein --sift --summary --symbol --tsl --uniprot --xref_refseq
--summary	Output only a comma-separated list of all observed consequences per variant. Transcript-specific columns will be left blank. *Not used by default*		--appris --biotype --canonical --ccds --coding_only --domains --flag_pick --flag_pick_allele --most_severe --no_intergenic --numbers --pick --pick_allele --polyphen --protein --sift --symbol --tsl --uniprot --xref_refseq
--filter_common	Shortcut flag for the filters below - this will exclude variants that have a co-located existing variant with global AF > 0.01 (1%). May be modified using any of the following freq_* filters. *Not used by default*	FREQS	
--check_frequency	Turns on frequency filtering. Use this to include or exclude variants based on the frequency of co-located existing variants in the Ensembl Variation database. You must also specify all	FREQS	

	of the --freq_* flags below. Frequencies used in filtering are added to the output under the FREQS key in the Extra field. *Not used by default*
`--freq_pop [pop]`	Name of the population to use in frequency filter. This must be one of the following: **Name** / **Description** 1KG_ALL — 1000 genomes combined population (global) 1KG_AFR — 1000 genomes combined African population 1KG_AMR — 1000 genomes combined American population 1KG_EAS — 1000 genomes combined East Asian population 1KG_EUR — 1000 genomes combined European population 1KG_SAS — 1000 genomes combined South Asian population AA — NHLBI-ESP African American EA — NHLBI-ESP European American gnomAD — gnomAD combined population gnomAD_AFR — gnomAD African/African American population gnomAD_AMR — gnomAD Latino population gnomAD_ASJ — gnomAD Ashkenazi Jewish population gnomAD_EAS — gnomAD East Asian population gnomAD_FIN — gnomAD Finnish population gnomAD_NFE — gnomAD non-Finnish European population gnomAD_OTH — gnomAD other population gnomAD_SAS — gnomAD South Asian population
`--freq_freq [freq]`	Allele frequency to use for filtering. Must be a float value between 0 and 1
`--freq_gt_lt [gt|lt]`	Specify whether the frequency of the co-located variant must be greater than (**gt**) or less than (**lt**) the value specified with --freq_freq
`--freq_filter [exclude|include]`	Specify whether to **exclude** or **include** only variants that pass the frequency filter

Database options

Flag	Alternate	Description	Output fields	Incompatible with
`--database`		Enable VEP to use local or remote databases.		--af_1kg --af_esp --af_exac --af_gnomad --cache --max_af --offline --pubmed --var_synonyms
`--host [hostname]`		Manually define the database host to connect to. Users in the US may find connection and transfer speeds quicker using our East coast mirror, useastdb.ensembl.org. *Default = "ensembldb.ensembl.org"*		
`--user [username]`	`-u`	Manually define the database username. *Default = "anonymous"*		
`--password [password]`	`--pass`	Manually define the database password. *Not used by default*		
`--port [number]`		Manually define the database port. *Default = 5306*		
`--genomes`		Override the default connection settings with those for the Ensembl Genomes public MySQL server. Required when using any of the Ensembl Genomes species. *Not used by default*		
`--is_multispecies [0|1]`		Some of the Ensembl Genomes databases (mainly bacteria and protists) are composed of a collection of close species. It updates the database connection settings (i.e. the database name) if the value is set to 1. *Default: 0*		
`--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 used by default*		--offline
`--db_version [number]`		Force VEP to connect to a specific version of the Ensembl databases. Not recommended as there may be conflicts between software and database versions. *Not used by default*		
`--registry [filename]`		Defining a registry file overwrites other connection settings and uses those found in the specified registry file to connect. *Not used by default*		

Variant Effect Predictor — Annotation sources

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

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

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

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

Caches

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

Downloading caches

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

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

Manually downloading caches

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

- **Indexed cache** (/release-103/variation/indexed_vep_cache/) - requires Bio::DB::HTS (setup by INSTALL.pl) or tabix, e.g.:

```
cd $HOME/.vep
curl -O /release-103/variation/indexed_vep_cache/homo_sapiens_vep_103_GRCh38.tar.gz
tar xzf homo_sapiens_vep_103_GRCh38.tar.gz
```

- **Non-indexed cache** (/release-103/variation/vep/), e.g.:

```
cd $HOME/.vep
curl -O /release-103/variation/vep/homo_sapiens_vep_103_GRCh38.tar.gz
tar xzf homo_sapiens_vep_103_GRCh38.tar.gz
```

FTP directories by species grouping:

Ensembl:	Vertebrates (indexed) | Vertebrates
Ensembl Genomes:	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.

Data in the cache

The data content of VEP caches vary by species. This table shows the contents of the default human cache files in release 103.

Source	Version (GRCh38)	Version (GRCh37)
Ensembl database version	103	103
Genome assembly	GRCh38.p13	GRCh37.p13
GENCODE	37	19
RefSeq	2020-09-29 (GCF_000001405.39_GRCh38.p13_genomic.gff)	2019-11-01 (GCF_000001405.25_GRCh37.p13_genomic.gff)
Regulatory build	1.0	1.0
PolyPhen	2.2.2	2.2.2
SIFT	5.2.2	5.2.2
dbSNP	154	153
COSMIC	92	90
HGMD-PUBLIC	2019.4	2019.4
ClinVar	2020-08	2019-12
1000 Genomes	Phase 3 (remapped)	Phase 3
NHLBI-ESP	V2-SSA137 (remapped)	V2-SSA137
gnomAD	r2.1, exomes only (remapped)	r2.1, exomes only

Convert with tabix

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

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

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

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

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

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

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

Data privacy and offline mode

When using the public database servers, VEP requests transcript and variation data that overlap the loci in your input file. As such, these coordinates are transmitted over the network to a public server, which may not be appropriate for the analysis of sensitive or private data.

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

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

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

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

GFF/GTF files

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

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

```
grep -v "#" data.gff | sort -k1,1 -k4,4n -k5,5n -t$'\t' | bgzip -c > data.gff.gz
tabix -p gff data.gff.gz
./vep -i input.vcf --gff data.gff.gz --fasta genome.fa.gz
```

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

- **GFF file**
  Example of command line with GFF, using of flag --gff :

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

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

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

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

- **GTF file**
  Example of command line with GTF, using of flag --gtf :

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

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

  ```
  --custom data.gtf.gz,,gtf
  ```

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

GFF format expectations

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

Column "type" (3rd column):

The following entity/feature types are supported by VEP. Lines of other types will be ignored; if this leads to an incomplete transcript model, the whole transcript model may be discarded.

- aberrant_processed_transcript
- CDS
- C_gene_segment
- D_gene_segment
- exon
- gene
- J_gene_segment
- lincRNA
- lincRNA_gene
- miRNA
- miRNA_gene
- mRNA
- processed_pseudogene
- processed_transcript
- pseudogene
- pseudogenic_transcript
- RNA
- rRNA
- rRNA_gene
- snoRNA
- snoRNA_gene
- snRNA
- snRNA_gene
- supercontig

- mt_gene
- ncRNA
- NMD_transcript_variant
- primary_transcript
- transcript
- tRNA
- VD_gene_segment
- V_gene_segment

Expected parameters in the 9th column:

- **ID**
  Only required for the genes and transcripts entities.
- **parent/Parent**
  - Entities in the GFF are expected to be linked using a key named "**parent**" or "**Parent**" in the attributes (9th) column of the GFF.
  - Unlinked entities (i.e. those with no parents **or** children) are discarded.
  - Sibling entities (those that share the same parent) may have overlapping coordinates, e.g. for exon and CDS entities.
- **biotype**
  Transcripts require a Sequence Ontology biotype to be defined in order to be parsed by VEP.
  The simplest way to define this is using an attribute named "**biotype**" on the transcript entity. Other configurations are supported in order for VEP to be able to parse GFF files from NCBI and other sources.

Here is an example:

```
##gff-version 3.2.1
##sequence-region 1 1 10000
1 Ensembl gene        1000  5000  . + . ID=gene1;Name=GENE1
1 Ensembl transcript  1100  4900  . + . ID=transcript1;Name=GENE1-001;Parent=gene1;biotype=protein_coding
1 Ensembl exon        1200  1300  . + . ID=exon1;Name=GENE1-001_1;Parent=transcript1
1 Ensembl exon        1500  3000  . + . ID=exon2;Name=GENE1-001_2;Parent=transcript1
1 Ensembl exon        3500  4000  . + . ID=exon3;Name=GENE1-001_2;Parent=transcript1
1 Ensembl CDS         1300  3800  . + . ID=cds1;Name=CDS0001;Parent=transcript1
```

GTF format expectations

The following GTF entity types will be extracted:

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

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

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

Here is an example:

```
1 Ensembl gene        1000  5000  . + . gene_id "gene1"; gene_name "GENE1";
1 Ensembl transcript  1100  4900  . + . gene_id "gene1"; transcript_id "transcript1"; gene_name "GENE1"; transcript_name "GENE1-001"; transc
1 Ensembl exon        1200  1300  . + . gene_id "gene1"; transcript_id "transcript1"; exon_number "exon1"; exon_id "GENE1-001_1";
1 Ensembl exon        1500  3000  . + . gene_id "gene1"; transcript_id "transcript1"; exon_number "exon2"; exon_id "GENE1-001_2";
1 Ensembl exon        3500  4000  . + . gene_id "gene1"; transcript_id "transcript1"; exon_number "exon3"; exon_id "GENE1-001_2";
1 Ensembl CDS         1300  3800  . + . gene_id "gene1"; transcript_id "transcript1"; exon_number "exon2"; ccds_id "CDS0001";
```

Chromosome synonyms

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

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

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

```
./vep -i input.vcf -cache -gff data.gff.gz -fasta genome.fa.gz -synonyms ~/.vep/homo_sapiens/103_GRCh38/chr_synonyms.txt
```

FASTA files

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

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

To enable this VEP uses one of two modules:

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

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

FASTA FTP directories

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

```
curl -O /release-103/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
gzip -d Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
bgzip Homo_sapiens.GRCh38.dna.primary_assembly.fa
./vep -i input.vcf --offline --hgvs --fasta Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
```

Databases

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

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

Public database servers

By default, VEP is configured to connect to the public Ensembl MySQL instance at ensembldb.ensembl.org. If you are in the USA (or geographically closer to the east coast of the USA than to the Ensembl 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`

Data for Ensembl Genomes species (e.g. plants, fungi, microbes) is available through a different public MySQL server. The appropriate connection parameters can be automatically loaded by using the flag --genomes

If you have a very small data set (100s of variants), using the public database servers should provide adequate performance. If you have larger data sets, or wish to use VEP in a batch manner, consider one of the alternatives below.

Using a local database

It is possible to set up a local MySQL mirror with the databases for your species of interest installed. For instructions on installing a local mirror, see here. You will need a MySQL server that you can connect to from the machine where you will run VEP (this can be the same machine). For most of the functionality of VEP, you will only need the Core database (e.g. homo_sapiens_core_103_38) installed. In order to find co-located variants or to use SIFT or PolyPhen, it is also necessary to install the relevant variation database (e.g. homo_sapiens_variation_103_38).

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

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

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

Bio::EnsEMBL::DBSQL::DBAdaptor->new(
  '-species' => "Homo_sapiens",
  '-group'   => "core",
  '-port'    => 5306,
  '-host'    => 'ensembldb.ensembl.org',
  '-user'    => 'anonymous',
  '-pass'    => '',
  '-dbname'  => 'homo_sapiens_core_103_38'
);

Bio::EnsEMBL::Variation::DBSQL::DBAdaptor->new(
  '-species' => "Homo_sapiens",
  '-group'   => "variation",
  '-port'    => 5306,
  '-host'    => 'ensembldb.ensembl.org',
  '-user'    => 'anonymous',
  '-pass'    => '',
  '-dbname'  => 'homo_sapiens_variation_103_38'
);

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

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

Cache - technical information

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

The following hash keys are deleted from each transcript object:

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

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

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

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

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

```
$transcript->translateable_seq
```

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

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

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

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

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

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

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

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

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

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

Variant Effect Predictor 🔍 Filtering results

The VEP package includes a tool, filter_vep, to filter results files on a variety of attributes.

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

Running filter_vep

Run as follows:

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

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

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

The above command removes known variants from the 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, filter_vep will attempt to read from STDIN. Input may be gzipped - to read a gzipped file use **`--gz`**
`--format [format]`		Specify input file format: • **tab** (i.e. the VEP results file) • **vcf**
`--output_file [file]`	`-o`	Specify the output file to write to. If no output file is specified, the filter_vep will write to STDOUT
`--force_overwrite`		Force an output file of the same name to be overwritten
`--filter [filters]`	`-f`	Add filter (see below). Multiple **`--filter`** flags may be used, and are treated as logical ANDs, i.e. all filters must pass for a line to be printed
`--soft_filter`		Variants not passing given filters will be flagged in the FILTER column of the VCF file, and will not be removed from output.
`--list`	`-l`	List allowed fields from the input file
`--count`	`-c`	Print only a count of matched lines
`--only_matched`		In VCF files, the CSQ field that contains the consequence data will often contain more than one "block" of consequence data, where each block corresponds to a variant/feature overlap. Using **`--only_matched`** will remove blocks that do not pass the filters. By default, filter_vep prints out the entire VCF line if any of the blocks pass the filters.
`--vcf_info_field [key]`		With VCF input files, by default filter_vep expects to find VEP annotations encoded in the CSQ INFO key; VEP itself can be configured to write to a different key (with the equivalent `--vcf_info_field` flag). Use this flag to change the INFO key VEP expects to decode: e.g. use the command "**`--vcf_info_field ANN`**" if the VEP annotations are stored in the INFO key "ANN".
`--ontology`	`-y`	Use Sequence Ontology to match consequence terms. Use with operator "is" to match against all child terms of your value. e.g. "Consequence is coding_sequence_variant" will match missense_variant, synonymous_variant etc. Requires database connection; defaults to connecting to ensembldb.ensembl.org. Use **`--host`**, **`--port`**, **`--user`**, **`--password`**, **`--version`** as per **`vep`** to change connection parameters.

Writing filters

Filter strings consist of three components:

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

Examples:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Operators

- **is** (synonyms: = , eq) : Match exactly

  ```
  # get only transcript consequences
  --filter "Feature_type is Transcript"
  ```

- **!=** (synonym: ne) : Does not match exactly

  ```
  # filter out tolerated SIFT predictions
  --filter "SIFT != tolerated"
  ```

- **match** (synonyms: matches , re , regex) : Match string using regular expression. You may include any regular expression notation, e.g. "\d" for any numerical character

  ```
  # match stop_gained, stop_lost and stop_retained
  --filter "Consequence match stop"
  ```

- **<** (synonym: lt) : Less than. Note an absent value is not considered to be equivalent to 0.

  ```
  # find SIFT scores less than 0.1
  --filter "SIFT < 0.1"
  ```

- **>** (synonym: gt) : Greater than

  ```
  # find variants not in the first exon
  --filter "Exon > 1"
  ```

- **<=** (synonym: lte) : Less than or equal to. Note an absent value is not considered to be equivalent to 0.
- **>=** (synonym: gte) : Greater than or equal to
- **exists** (synonyms: ex , defined) : Field is defined - equivalent to using no operator and value
- **in** : Find in list or file. Value may be either a comma-separated list or a file containing values on separate lines. Each list item is compared using the "is" operator.

  ```
  # find variants in a list of gene names
  --filter "SYMBOL in BRCA1,BRCA2"

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

Variant Effect Predictor — Custom annotations

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

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

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

Data formats

VEP supports the following formats:

- **Gene/transcript annotations**
  - GFF : a format for describing genes and other genomic features — format specifications.
  - GTF : a similar format derived from GFF — format specifications.

  See more documentation about GFF/GTF format requirements for VEP.

  NOTE: It requires a **FASTA** file on the offline mode.

- **Variant data**
  - VCF : a format used to describe genomic variants. VEP will use the 3rd column of the file as the identifier. INFO fields from records may be added to the VEP output.

- **Basic/uninterpreted data**
  - BED : a simple tab-delimited format containing 3-12 columns of data. The first 3 columns contain the coordinates of the feature. If available, VEP will use the 4th column of the file as the identifier of the feature.
  - bigWig : a format for storage of dense continuous data. VEP uses the value for the given position as the "identifier". Note that bigWig files contain their own indices, and do not need to be indexed by tabix. Requires Bio::DB::BigFile.

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

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

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

Preparing files

Custom annotation files must be prepared in a particular way in order to work with tabix and therefore with VEP. Files must be stripped of comment lines, sorted in chromosome and position order, compressed using bgzip and finally indexed using tabix. Here are some examples of that process for:

- **GFF file**

```
grep -v "#" myData.gff | sort -k1,1 -k4,4n -k5,5n -t$'\t' | bgzip -c > myData.gff.gz
tabix -p gff myData.gff.gz
```

- **BED file**

```
grep -v "#" myData.bed | sort -k1,1 -k2,2n -k3,3n -t$'\t' | bgzip -c > myData.bed.gz
tabix -p bed myData.bed.gz
```

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

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

Options

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

```
./vep [...] --custom Filename , Short_name , File_type , Annotation_type , Force_report_coordinates , VCF_fields
```

The options are as follows:

- **Filename** :

  The path to the file. For tabix indexed files, the VEP will check that both the file and the corresponding .tbi file exist. For remote files, VEP will check that the tabix index is accessible on startup.

- **Short name** :

  A name for the annotation that will appear as the key in the key=value pairs in the results.
  If not defined, this will default to the annotation filename for the first set of annotation added (e.g. "myPhenotypes.bed.gz" in the second example below if the short name was missing).

- **File type** :

  `"bed", "gff", "gtf", "vcf" or "bigwig"`

- **Annotation type** :

  `"exact" or "overlap"` (if left blank, assumed to be **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** :

`"0" or "1"` (if left blank, assumed to be **0**)

If set to "1", this forces VEP to output the coordinates of an overlapping custom feature instead of any found identifier (or value in the case of bigWig) field. If set to "0" (the default), VEP will output the identifier field if one is found; if none is found, then the coordinates are used instead.

- **VCF fields** :

  You can specify any info type (e.g. "AC") present in the INFO field of the custom input VCF, to add these as custom annotations:

  - If using "exact" annotation type, allele-specific annotation will be retrieved.
  - The INFO field name will be prefixed with the short name, e.g. using short name "test", the INFO field "foo" will appear as "test_FOO" in the VEP output.
  - In VCF files the custom annotations are added to the CSQ INFO field.
  - Alleles in the input and VCF entry are trimmed in both directions in an attempt to match complex or poorly formatted entries.

For example:

```
# BigWig file
./vep [...] --custom frequencies.bw,Frequency,bigwig,exact,0
# BED file
./vep [...] --custom http://www.myserver.com/data/myPhenotypes.bed.gz,Phenotype,bed,exact,1
# VCF file
./vep [...] --custom [[SPECIES_DEFS::ENSEMBL_FTP_URL]]/data_files/homo_sapiens/GRCh37/variation_genotype/TOPMED_GRCh37.vcf.gz,,vcf,exact,0,
```

Example - ClinVar

We include the most recent public variant and phenotype data available in each Ensembl release, but some projects release data more frequently than we do.
If you want to have the very latest annotations, you can use the data files from your prefered projects (in any format listed in Data formats) and use them as a VEP custom annotation.

For instance, you can annotate you variants with VEP, using the the latest ClinVar data as custom annotation.
ClinVar provides VCF files on their FTP site: GRCh37 and GRCh38.

See below an example about how to use ClinVar VCF files as a VEP custom annotation:

1. Download the VCF files (you need the compressed VCF file and the index file), e.g.:

```
# Compressed VCF file
curl -O ftp://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz
# Index file
curl -O ftp://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz.tbi
```

2. Example of command you can use:

```
./vep [...] --custom clinvar.vcf.gz,ClinVar,vcf,exact,0,CLNSIG,CLNREVSTAT,CLNDN

## Where the selected ClinVar INFO fields (from the ClinVar VCF file) are:
# - CLNSIG:      Clinical significance for this single variant
# - CLNREVSTAT: ClinVar review status for the Variation ID
# - CLNDN:       ClinVar's preferred disease name for the concept specified by disease identifiers in CLNDISDB
# Of course you can select the INFO fields you want in the ClinVar VCF file

# Quick example on GRCh38:
./vep --id "1  230710048 230710048 A/G 1" --species homo_sapiens -o /path/to/output/output.txt --cache --offline --assembly GRCh38 --cus
```

Results in the default VEP format

```
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Feature_type : Type of feature - Transcript, RegulatoryFeature or MotifFeature
## Consequence : Consequence type
## cDNA_position : Relative position of base pair in cDNA sequence
## CDS_position : Relative position of base pair in coding sequence
## Protein_position : Relative position of amino acid in protein
## Amino_acids : Reference and variant amino acids
## Codons : Reference and variant codon sequence
## Existing_variation : Identifier(s) of co-located known variants
## Extra column keys:
## IMPACT : Subjective impact classification of consequence type
## DISTANCE : Shortest distance from variant to transcript
## STRAND : Strand of the feature (1/-1)
## FLAGS : Transcript quality flags
## SOURCE : Source of transcript
## ClinVar : /opt/vep/.vep/custom/clinvar.vcf.gz (exact)
## ClinVar_CLNSIG : CLNSIG field from /path/to/custom_files/clinvar.vcf.gz
## ClinVar_CLNREVSTAT : CLNREVSTAT field from /path/to/custom_files/clinvar.vcf.gz
## ClinVar_CLNDN : CLNDN field from /path/to/custom_files/clinvar.vcf.gz
#Uploaded_variation  Location     Allele  Gene             Feature          Feature_type  Consequence              ...  Extra
1_230710048_A/G      1:230710048  G       ENSG00000135744  ENST00000366667  Transcript    missense_variant         ...  IMPACT=MODERATE;
1_230710048_A/G      1:230710048  G       ENSG00000244137  ENST00000412344  Transcript    downstream_gene_variant  ...  IMPACT=MODIFIER;
```

Results in VCF (adding the tag --vcf in the command line)

```
##fileformat=VCFv4.1
##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from Ensembl VEP. Format: Allele|Consequence|IMPACT|SYMBOL|Gene
##INFO=<ID=ClinVar,Number=.,Type=String,Description="/path/to/custom_files/clinvar.vcf.gz (exact)">
```

```
##INFO=<ID=ClinVar_CLNSIG,Number=.,Type=String,Description="CLNSIG field from /path/to/custom_files/clinvar.vcf.gz">
##INFO=<ID=ClinVar_CLNREVSTAT,Number=.,Type=String,Description="CLNREVSTAT field from /path/to/custom_files/clinvar.vcf.gz">
##INFO=<ID=ClinVar_CLNDN,Number=.,Type=String,Description="CLNDN field from /path/to/custom_files/clinvar.vcf.gz">
#CHROM  POS        ID              REF  ALT  QUAL  FILTER  INFO
1       230710048  1_230710048_A/G  A    G    .     .       CSQ=G|missense_variant|MODERATE|AGT|ENSG00000135744|Transcript|ENST000003666
```

Using remote files

The tabix utility makes it possible to read annotation files from remote locations, for example over HTTP or FTP protocols.

In order to do this, the .tbi index file is downloaded locally (to the current working directory) when VEP is run. From this point on, only the portions of data requested by VEP (i.e. those overlapping the variants in your input file) are downloaded.

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

Variant Effect Predictor Plugins

VEP can use plugin modules written in Perl to **add functionality** to the software.

Plugins are a powerful way to **extend**, **filter** and **manipulate** the VEP output.
They can be installed using VEP's installer script, run the following command to get a list of available plugins:

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

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

Existing plugins

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

https://github.com/Ensembl/VEP_plugins

Here is the list of the VEP plugins available:

Select categories: All categories

Plugin	Description
AncestralAllele	A VEP plugin that retrieves ancestral allele sequences from a FASTA file. ... Ensembl produces FASTA file dumps of the ancestral sequences of key species. They are available from [[SPECIES_DEFS::ENSEMBL_FTP_URL]]/current_fasta/ancestral_alleles/ For optimal retrieval speed, you should pre-process the FASTA files into a single bgzipped file that can be accessed via Bio::DB::HTS::Faidx (installed by VEP's wget [[SPECIES_DEFS::ENSEMBL_FTP_URL]]/current_fasta/ancestral_alleles/homo_sapiens_ancestor_GRCh38.tar.gz tar xfz homo_sapiens_ancestor_GRCh38.tar.gz cat homo_sapiens_ancestor_GRCh38/*.fa | bgzip -c > homo_sapiens_ancestor_GRCh38.fa.gz rm -rf homo_sapiens_ancestor_GRCh38/ homo_sapiens_ancestor_GRCh38.tar.gz `./vep -i variations.vcf --plugin AncestralAllele,homo_sapiens_ancestor_GRCh38.fa.gz` The plugin is also compatible with Bio::DB::Fasta and an uncompressed FASTA file. Note the first time you run the plugin with a newly generated FASTA file it will spend some time indexing the file. DO NOT INTERRUPT THIS PROCESS, particu not have Bio::DB::HTS installed. Special cases: "-" represents an insertion "?" indicates the chromosome could not be looked up in the FASTA
Blosum62	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that looks up the BLOSUM 62 substitution matrix score for the reference and alternative amino acids predicted for a missense mutation. It adds one new entry to the VEP's Extra column, BLOSUM62 which is the associated score.
CADD Combined Annotation Dependent Depletion	A VEP plugin that retrieves CADD scores for variants from one or more tabix-indexed CADD data files. ... Please cite the CADD publication alongside the VEP if you use this resource: https://www.ncbi.nlm.nih.gov/pubmed/24487276 The tabix utility must be installed in your path to use this plugin. The CADD data files can be downloaded from http://cadd.gs.washington.edu/download The plugin works with all versions of available CADD files. The plugin only reports scores and does not consider any additional annotations from a CADD file. It is therefore sufficient to use CADD files without the additional annotations.

Plugin	Description
Carol	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that calculates the Combined Annotation scoRing toOL (CAROL) score (1) for a missense mutation based on the pre-calculated SIFT (2) and PolyPhen-2 (3) scores from the Ensembl API (4). It adds one new entry class to the VEP's Extra column, CAROL which is the calculated CAROL score. Note that this module is a perl reimplementation of the original R script, available at: ... http://www.sanger.ac.uk/resources/software/carol/ I believe that both versions implement the same algorithm, but if there are any discrepancies the R version should be treated as the reference implementation. Bug reports are welcome. References: (1) Lopes MC, Joyce C, Ritchie GRS, John SL, Cunningham F, Asimit J, Zeggini E. A combined functional annotation score for non-synonymous variants Human Heredity (in press) (2) Kumar P, Henikoff S, Ng PC. Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm Nature Protocols 4(8):1073-1081 (2009) doi:10.1038/nprot.2009.86 (3) Adzhubei IA, Schmidt S, Peshkin L, Ramensky VE, Gerasimova A, Bork P, Kondrashov AS, Sunyaev SR. A method and server for predicting damaging missense mutations Nature Methods 7(4):248-249 (2010) doi:10.1038/nmeth0410-248 (4) Flicek P, et al. Ensembl 2012 Nucleic Acids Research (2011) doi: 10.1093/nar/gkr991
Condel	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that calculates the Consensus Deleteriousness (Condel) score (1) for a missense mutation based on the pre-calculated SIFT (2) and PolyPhen-2 (3) scores from the Ensembl API (4). It adds one new entry class to the VEP's Extra column, Condel which is the calculated Condel score. This version of Condel was developed by the Biomedical Genomics Group of the Universitat Pompeu Fabra, at the Barcelona Biomedical Research Park and available at (http://bg.upf.edu/condel) until April 2014. The code in this plugin is based on a script provided by this group and slightly reformatted to fit into the Ensembl API. ... The plugin takes 3 command line arguments, the first is the path to a Condel configuration directory which contains cutoffs and the distribution files etc., the second is either "s", "p", or "b" to output the Condel score, prediction or both (the default is both), and the third argument is either 1 or 2 to use the original version of Condel (1), or the newer version (2) - 2 is the default and is recommended to avoid false positive predictions from Condel in some circumstances. An example Condel configuration file and a set of distribution files can be found in the config/Condel directory in this repository. You should edit the config/Condel/config/condel_SP.conf file and set the 'condel.dir' parameter to the full path to the location of the config/Condel directory on your system. References: (1) Gonzalez-Perez A, Lopez-Bigas N. Improving the assessment of the outcome of non-synonymous SNVs with a Consensus deleteriousness score (Condel) Am J Hum Genet 88(4):440-449 (2011) doi:10.1016/j.ajhg.2011.03.004 (2) Kumar P, Henikoff S, Ng PC. Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm Nature Protocols 4(8):1073-1081 (2009) doi:10.1038/nprot.2009.86 (3) Adzhubei IA, Schmidt S, Peshkin L, Ramensky VE, Gerasimova A, Bork P, Kondrashov AS, Sunyaev SR. A method and server for predicting damaging missense mutations Nature Methods 7(4):248-249 (2010) doi:10.1038/nmeth0410-248 (4) Flicek P, et al. Ensembl 2012 Nucleic Acids Research (2011) doi: 10.1093/nar/gkr991
Conservation	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that retrieves a conservation score from the Ensembl Compara databases for variant positions. You can specify the method link type and species sets as command line options, the default is to fetch GERP scores from the EPO 35 way mammalian alignment (please refer to the Compara documentation for more details of available analyses). ... If a variant affects multiple nucleotides the average score for the position will be returned, and for insertions the average score of the 2 flanking bases will be returned. The plugin uses the ensembl-compara API module (optional, see http://www.ensembl.org/info/docs/api/index.html) or obtains data directly from BigWig files (optional, see [[SPECIES_DEFS::ENSEMBL_FTP_URL]]/current_compara/conservation_scores/)

Plugin	Description

dbNSFP	A VEP plugin that retrieves data for missense variants from a tabix-indexed dbNSFP file. ...
	
	Please cite the dbNSFP publications alongside the VEP if you use this resource:
	dbNSFP https://www.ncbi.nlm.nih.gov/pubmed/21520341
	dbNSFP v2.0 https://www.ncbi.nlm.nih.gov/pubmed/23843252
	dbNSFP v3.0 https://www.ncbi.nlm.nih.gov/pubmed/26555599
	
	You must have the Bio::DB::HTS module or the tabix utility must be installed in your path to use this plugin. The dbNSFP data file can be downloaded from https://sites.google.com/site/jpopgen/dbNSFP.
	
	Release 3.5a of dbNSFP uses GRCh38/hg38 coordinates and GRCh37/hg19 coordinates.
	To use the plugin with GRCh37/hg19 data:
	> wget ftp://dbnsfp:dbnsfp@dbnsfp.softgenetics.com/dbNSFPv3.5a.zip
	> unzip dbNSFPv3.5a.zip
	> head -n1 dbNSFP3.5a_variant.chr1 > h
	> cat dbNSFP3.5a_variant.chr* | grep -v ^#chr | awk '$8 != "."' | sort -k8,8 -k9,9n - | cat h - | bgzip -c > dbNSFP_hg19.gz
	> tabix -s 8 -b 9 -e 9 dbNSFP_hg19.gz
	
	To use the plugin with GRCh38/hg38 data:
	> wget ftp://dbnsfp:dbnsfp@dbnsfp.softgenetics.com/dbNSFPv3.5a.zip
	> unzip dbNSFPv3.5a.zip
	> head -n1 dbNSFP3.5a_variant.chr1 > h
	> cat dbNSFP3.5a_variant.chr* | grep -v ^#chr | sort -k1,1 -k2,2n - | cat h - | bgzip -c > dbNSFP.gz
	> tabix -s 1 -b 2 -e 2 dbNSFP.gz
	
	For release 4.0a with GRCh38/hg38 data:
	> wget ftp://dbnsfp:dbnsfp@dbnsfp.softgenetics.com/dbNSFP4.0a.zip
	> unzip dbNSFP4.0a.zip
	> zcat dbNSFP4.0a_variant.chr1.gz | head -n1 > h
	> zgrep -h -v ^#chr dbNSFP4.0a_variant.chr* | sort -k1,1 -k2,2n - | cat h - | bgzip -c > dbNSFP4.0a.gz
	> tabix -s 1 -b 2 -e 2 dbNSFP4.0a.gz
	
	When running the plugin you must list at least one column to retrieve from the dbNSFP file, specified as parameters to the plugin e.g.
	
	`--plugin dbNSFP,/path/to/dbNSFP.gz,LRT_score,GERP++_RS`
	
	You may include all columns with ALL; this fetches a large amount of data per variant!:
	
	`--plugin dbNSFP,/path/to/dbNSFP.gz,ALL`
	
	Tabix also allows the data file to be hosted on a remote server. This plugin is fully compatible with such a setup - simply use the URL of the remote file:
	
	`--plugin dbNSFP,http://my.files.com/dbNSFP.gz,col1,col2`
	
	The plugin replaces occurrences of ';' with ',' and 'l' with '>'. However, some data field columns, e.g. Interpro_domain, use the replacement characters. We addec replacement logic for customising the required replacement of ';' and 'l' in dbNSFP data columns. In addition to the default replacements (; to , and l to >) users c customised replacements. Users can either modify
	the file dbNSFP_replacement_logic in the VEP_plugins directory or provide their own
	file as second argument when calling the plugin:
	
	`--plugin dbNSFP,/path/to/dbNSFP.gz,/path/to/dbNSFP_replacement_logic,LRT_score,GERP++_RS`
	
	Note that transcript sequences referred to in dbNSFP may be out of sync with those in the latest release of Ensembl; this may lead to discrepancies with scores retrieved from other sources.
	
	If the dbNSFP README file is found in the same directory as the data file, column descriptions will be read from this and incorporated into the VEP output file header.

Plugin	Description
dbscSNV	A VEP plugin that retrieves data for splicing variants from a tabix-indexed dbscSNV file. ... Please cite the dbscSNV publication alongside the VEP if you use this resource: http://nar.oxfordjournals.org/content/42/22/13534 The Bio::DB::HTS perl library or tabix utility must be installed in your path to use this plugin. The dbscSNV data file can be downloaded from https://sites.google.com/site/jpopgen/dbNSFP. The file must be processed and indexed by tabix before use by this plugin. dbscSNV1.1 has coordinates for both GRCh38 and GRCh37; the file must be processed differently according to the assembly you use. > wget ftp://dbnsfp:dbnsfp@dbnsfp.softgenetics.com/dbscSNV1.1.zip > unzip dbscSNV1.1.zip > head -n1 dbscSNV1.1.chr1 > h # GRCh38 > cat dbscSNV1.1.chr* | grep -v ^chr | sort -k5,5 -k6,6n | cat h - | awk '$5 != "."' | bgzip -c > dbscSNV1.1_GRCh38.txt.gz > tabix -s 5 -b 6 -e 6 -c c dbscSNV1.1_GRCh38.txt.gz # GRCh37 > cat dbscSNV1.1.chr* | grep -v ^chr | cat h - | bgzip -c > dbscSNV1.1_GRCh37.txt.gz > tabix -s 1 -b 2 -e 2 -c c dbscSNV1.1_GRCh37.txt.gz Note that in the last command we tell tabix that the header line starts with "c"; this may change to the default of "#" in future versions of dbscSNV. Tabix also allows the data file to be hosted on a remote server. This plugin is fully compatible with such a setup - simply use the URL of the remote file: `--plugin dbscSNV,http://my.files.com/dbscSNV.txt.gz` Note that transcript sequences referred to in dbscSNV may be out of sync with those in the latest release of Ensembl; this may lead to discrepancies with scores retrieved from other sources.

Plugin	Description

DisGeNET	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds Variant-Disease-PMID associations from the DisGeNET database. It is available for GRCh38. ...
	
	Please cite the DisGeNET publication alongside the VEP if you use this resource: https://academic.oup.com/nar/article/48/D1/D845/5611674
	
	Options are passed to the plugin as key=value pairs:
	
	file : Path to DisGENET data file (mandatory).
	
	disease : Set value to 1 to include the diseases/phenotype names reporting the Variant-PMID association (optional).
	
	rsid : Set value to 1 to include the dbSNP variant Identifier (optional).
	
	filter_score : Only reports citations with score greater or equal than input value (optional).
	
	filter_source : Only reports citations from input sources (optional).
	Accepted sources are: UNIPROT, CLINVAR, GWASDB, GWASCAT, BEFREE
	Separate multiple values with '&'.
	
	unique : Only reports unique dbSNP variant Identifiers and diseases/phenotype names (optional)
	
	Output:
	The output includes:
	- PMID of the publication reporting the Variant-Disease association (default)
	- DisGENET score for the Variant-Disease association (default)
	- dbSNP variant Identifier (optional)
	- diseases/phenotype names (optional)
	
	The following steps are necessary before running this plugin (tested with DisGeNET export date 2020-05-26):
	This plugin uses file 'all_variant_disease_pmid_associations.tsv.gz'
	File can be downloaded from: https://www.disgenet.org/downloads
	
	gunzip all_variant_disease_pmid_associations.tsv.gz
	
	awk '($1 ~ /^snpId/ || $2 ~ /NA/) {next} {print $0}'
	all_variant_disease_pmid_associations.tsv > all_variant_disease_pmid_associations_clean.tsv
	
	sort -t $'\t' -k2,2 -k3,3n
	all_variant_disease_pmid_associations_clean.tsv > all_variant_disease_pmid_associations_sorted.tsv
	
	awk '{ gsub (/\t +/, "\t", $0); print}'
	all_variant_disease_pmid_associations_sorted.tsv > all_variant_disease_pmid_associations_final.tsv
	
	bgzip all_variant_disease_pmid_associations_final.tsv
	tabix -s 2 -b 3 -e 3 all_variant_disease_pmid_associations_final.tsv.gz
	
	The plugin can then be run as default:
	
	`./vep -i variations.vcf --plugin DisGeNET,file=all_variant_disease_pmid_associations_final.tsv.gz`
	
	or with an option to include optional data or/and filters:
	
	`./vep -i variations.vcf --plugin DisGeNET,file=all_variant_disease_pmid_associations_final.tsv.gz,`
	
	disease=1
	
	`./vep -i variations.vcf --plugin DisGeNET,file=all_variant_disease_pmid_associations_final.tsv.gz,`
	
	disease=1,filter_source='GWASDB&GWASCAT'
	
	Of notice: this plugin only matches the chromosome and the position in the chromosome, the alleles are not taken into account to append the DisGENET data. The rsid is provided (optional) in the output in order to help to filter the relevant data.

Downstream	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that predicts the downstream effects of a frameshift variant on the protein sequence of a transcript. It provides the predicted downstream protein sequence (including any amino acids overlapped by the variant itself), and the change in length relative to the reference protein. ... Note that changes in splicing are not predicted - only the existing translateable (i.e. spliced) sequence is used as a source of translation. Any variants with a splice site consequence type are ignored. If VEP is run in offline mode using the flag --offline, a FASTA file is required. See: https://www.ensembl.org/info/docs/tools/vep/script/vep_cache.html#fasta Sequence may be incomplete without a FASTA file or database connection.

Plugin	Description
Draw	A VEP plugin that draws pictures of the transcript model showing the variant location. Can take five optional paramters: ... 1) File name stem for images 2) Image width in pixels (default: 1000px) 3) Image height in pixels (default: 100px) 4) Transcript ID - only draw images for this transcript 5) Variant ID - only draw images for this variant e.g. `./vep -i variations.vcf --plugin Draw,myimg,2000,100` Images are written to [file_stem]_[transcript_id]_[variant_id].png Requires GD library installed to run.
ExAC	A VEP plugin that retrieves ExAC allele frequencies. ... Visit ftp://ftp.broadinstitute.org/pub/ExAC_release/current to download the latest ExAC VCF. Note that the currently available version of the ExAC data file (0.3) is only available on the GRCh37 assembly; therefore it can only be used with this plugin when using the VEP on GRCh37. See http://www.ensembl.org/info/docs/tools/vep/script/vep_other.html#assembly The tabix utility must be installed in your path to use this plugin. The plugin takes 3 command line arguments. Second and third arguments are not mandatory. If AC specified as second argument Allele counts per population will be included in output. If AN specified as third argument Allele specific chromosome counts will be included in output.
ExACpLI	A VEP plugin that adds the probabililty of a gene being loss-of-function intolerant (pLI) to the VEP output. ... Lek et al. (2016) estimated pLI using the expectation-maximization (EM) algorithm and data from 60,706 individuals from ExAC (http://exac.broadinstitute.org/about). The closer pLI is to 1, the more likely the gene is loss-of-function (LoF) intolerant. Note: the pLI was calculated using a representative transcript and is reported by gene in the plugin. The data for the plugin is provided by Kaitlin Samocha and Daniel MacArthur. See https://www.ncbi.nlm.nih.gov/pubmed/27535533 for a description of the dataset and analysis. The ExACpLI_values.txt file is found alongside the plugin in the VEP_plugins GitHub repository. The file contains the fields gene and pLI extracted from the file at ftp://ftp.broadinstitute.org/pub/ExAC_release/release0.3/functional_gene_constraint/fordist_cleaned_exac_r03_march16_z_pli_rec_null_data.txt To use another values file, add it as a parameter i.e. `./vep -i variants.vcf --plugin ExACpLI,values_file.txt`
FATHMM	A VEP plugin that gets FATHMM scores and predictions for missense variants. ... You will need the fathmm.py script and its dependencies (Python, Python MySQLdb). You should create a "config.ini" file in the same directory as the fathmm.py script with the database connection options. More information about how to set up FATHMM can be found on the FATHMM website at https://github.com/HAShihab/fathmm. A typical installation could consist of: > wget https://raw.github.com/HAShihab/fathmm/master/cgi-bin/fathmm.py > wget http://fathmm.biocompute.org.uk/database/fathmm.v2.3.SQL.gz > gunzip fathmm.v2.3.SQL.gz > mysql -h[host] -P[port] -u[user] -p[pass] -e"CREATE DATABASE fathmm" > mysql -h[host] -P[port] -u[user] -p[pass] -Dfathmm < fathmm.v2.3.SQL > echo -e "[DATABASE]\nHOST = [host]\nPORT = [port]\nUSER = [user]\nPASSWD = [pass]\nDB = fathmm\n" > config.ini
FATHMM_MKL	A VEP plugin that retrieves FATHMM-MKL scores for variants from a tabix-indexed FATHMM-MKL data file. ... See https://github.com/HAShihab/fathmm-MKL for details. NB: The currently available data file is for GRCh37 only.
FlagLRG	A VEP plugin that retrieves the LRG ID matching either the RefSeq or Ensembl transcript IDs. You can obtain the 'list_LRGs_transcripts_xrefs.txt' using: ... > wget ftp://ftp.ebi.ac.uk/pub/databases/lrgex/list_LRGs_transcripts_xrefs.txt

Plugin	Description
FunMotifs	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds tissue-specific transcription factor motifs from FunMotifs to VEP output. ... Please cite the FunMotifs publication alongside the VEP if you use this resource. The preprint can be found at: https://www.biorxiv.org/content/10.1101/683722v1 FunMotifs files can be downloaded from: http://bioinf.icm.uu.se:3838/funmotifs/ At the time of writing, all BED files found through this link support GRCh37, however other assemblies are supported by the plugin if an appropriate BED file is supplied. The tabix utility must be installed in your path to use this plugin.

G2P
gene2phenotype
	A VEP plugin that uses G2P allelic requirements to assess variants in genes for potential phenotype involvement. ...
	
	The plugin has multiple configuration options, though minimally requires only the CSV file of G2P data.
	
	Options are passed to the plugin as key=value pairs, (defaults in parentheses):
	
	file : Path to G2P data file. The file needs to be uncompressed.
	- Download from http://www.ebi.ac.uk/gene2phenotype/downloads
	- Download from PanelApp
	
	variant_include_list : A list of variants to include even if variants do not pass allele frequency filtering. The include list needs to be a sorted, bgzipped and tabixed VCF file.
	
	af_monoallelic : maximum allele frequency for inclusion for monoallelic genes (0.0001)
	
	af_biallelic : maximum allele frequency for inclusion for biallelic genes (0.005)
	confidence_levels : Confidence levels to include: confirmed, probable, possible, both RD and IF.
	Separate multiple values with '&'.
	https://www.ebi.ac.uk/gene2phenotype/terminology
	Default levels are confirmed and probable.
	all_confidence_levels : Set value to 1 to include all confidence levels: confirmed, probable and possible.
	Setting the value to 1 will overwrite any confidence levels provided with the confidence_levels option.
	af_from_vcf : set value to 1 to include allele frequencies from VCF file.
	Specifiy the list of reference populations to include with --af_from_vcf_keys
	af_from_vcf_keys : VCF collections used for annotating variant alleles with observed allele frequencies. Allele frequencies are retrieved from VCF files. If af_from_vcf is set to 1 but no VCF collections are specified with --af_from_vcf_keys all available VCF collections are included.
	Available VCF collections: topmed, uk10k, gnomADe, gnomADg, gnomADg_r3.0
	Separate multiple values with '&'
	VCF collections contain the following populations:
	topmed: TOPMed
	uk10k: ALSPAC, TWINSUK
	gnomADe: gnomADe:AFR, gnomADe:ALL, gnomADe:AMR, gnomADe:ASJ, gnomADe:EAS, gnomADe:FIN, gnomADe:NFE, gnomADe:OTH, gnomADe:SAS
	gnomADg: gnomADg:AFR, gnomADg:ALL, gnomADg:AMR, gnomADg:ASJ, gnomADg:EAS, gnomADg:FIN, gnomADg:NFE, gnomADg:OTH
	default_af : default frequency of the input variant if no frequency data is found (0). This determines whether such variants are included; the value of 0 forces variants with no frequency data to be included as this is considered equivalent to having a frequency of 0. Set to 1 (or any value higher than af) to exclude them.
	types : SO consequence types to include. Separate multiple values with '&' (splice_donor_variant,splice_acceptor_variant,stop_gained, frameshift_variant,stop_lost,initiator_codon_variant, inframe_insertion,inframe_deletion,missense_variant, coding_sequence_variant,start_lost,transcript_ablation, transcript_amplification,protein_altering_variant)
	
	log_dir : write stats to log files in log_dir
	
	txt_report : write all G2P complete genes and attributes to txt file
	
	html_report : write all G2P complete genes and attributes to html file
	
	Example:
	
	`--plugin G2P,file=G2P.csv,af_monoallelic=0.05,types=stop_gained&frameshift_variant`
	
	`--plugin G2P,file=G2P.csv,af_monoallelic=0.05,af_from_vcf=1`
	
	`--plugin G2P,file=G2P.csv,af_from_vcf=1,af_from_vcf_keys=topmed&gnomADg`
	
	`--plugin G2P,file=G2P.csv,af_from_vcf=1,af_from_vcf_keys=topmed&gnomADg,confidence_levels='confirmed&probable&both R`
	
	`--plugin G2P,file=G2P.csv`

Plugin	Description
GeneSplicer	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that runs GeneSplicer (https://ccb.jhu.edu/software/genesplicer/) to get splice site predictions. ... It evaluates a tract of sequence either side of and including the variant, both in reference and alternate states. The amount of sequence included either side defaults to 100bp, but can be modified by passing e.g. "context=50" as a parameter to the plugin. Any predicted splicing regions that overlap the variant are reported in the output with one of four states: no_change, diff, gain, loss There follows a "/"-separated string consisting of the following data: 1) type (donor, acceptor) 2) coordinates (start-end) 3) confidence (Low, Medium, High) 4) score Example: loss/acceptor/727006-727007/High/16.231924 If multiple sites are predicted, their reports are separated by ",". For diff, the confidence and score for both the reference and alternate sequences is reported as REF-ALT. Example: diff/donor/621915-621914/Medium-Medium/7.020731-6.988368 Several parameters can be modified by passing them to the plugin string: context : change the amount of sequence added either side of the variant (default: 100bp) tmpdir : change the temporary directory used (default: /tmp) cache_size : change how many sequences' scores are cached in memory (default: 50) Example: --plugin GeneSplicer,$GS/bin/linux/genesplicer,$GS/human,context=200,tmpdir=/mytmp On some systems the binaries provided will not execute, but can be compiled from source: cd $GS/sources make cd - `./vep [options] --plugin GeneSplicer,$GS/sources/genesplicer,$GS/human` On Mac OSX the make step is known to fail; the genesplicer.cpp file requires modification: cd $GS/sources perl -pi -e "s/^main /int main /" genesplicer.cpp make
gnomADc	A VEP plugin that retrieves gnomAD annotation from either the genome or exome coverage files, available here: ... http://gnomad.broadinstitute.org/downloads The coverage files must be downloaded and tabix indexed before using this plugin: > release="2.0.2" > genomes="https://storage.googleapis.com/gnomad-public/release/${release}/coverage/genomes" > wget -x "${genomes}"/gnomad.genomes.r${release}.chr{{1..22},X}.coverage.txt.gz > for i in "${genomes#*//}"/gnomad.genomes.r${release}.chr{{1..22},X}.coverage.txt.gz; do > tabix -s 1 -b 2 -e 2 "${i}" > done > exomes="https://storage.googleapis.com/gnomad-public/release/${release}/coverage/exomes" > wget -x "${exomes}"/gnomad.exomes.r${release}.chr{{1..22},X,Y}.coverage.txt.gz > for i in "${exomes#*//}"/gnomad.exomes.r${release}.chr{{1..22},X,Y}.coverage.txt.gz; do > tabix -s 1 -b 2 -e 2 "${i}" > done The parent directory's basename is used to set the output field prefix. This is 'gnomADg' for genomes, 'gnomADe' for exomes, or else just 'gnomAD'. If you use this plugin, please see the terms and data information: http://gnomad.broadinstitute.org/terms The gnomAD coverage files are provided for GRCh37, but if you use GRCh38 you may like to use the liftover files, available here: https://console.cloud.google.com/storage/browser/gnomad-public/release You must have the Bio::DB::HTS module or the tabix utility must be installed in your path to use this plugin.
GO Gene Ontology	A VEP plugin that retrieves Gene Ontology terms associated with transcripts/translations via the Ensembl API. Requires database connection.
HGVSIntronOffset	A VEP plugin for the Ensembl Variant Effect Predictor (VEP) that returns HGVS intron start and end offsets. To be used with --hgvs option.

Plugin	Description

LD
Linkage Disequilibrium
	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that finds variants in linkage disequilibrium with any overlapping existing variants from the Ensembl variation databases. You can configure the population used to calculate the r2 value, and the r2 cutoff used by passing arguments to the plugin via the VEP command line (separated by commas). This plugin adds a single new entry to the Extra column with a comma-separated list of linked variant IDs and the associated r2 values, e.g.: ...
	
	LinkedVariants=rs123:0.879,rs234:0.943
	
	If no arguments are supplied, the default population used is the CEU sample from the 1000 Genomes Project phase 3, and the default r2 cutoff used is 0.8.
	
	WARNING: Calculating LD is a relatively slow procedure, so this will slow VEP down considerably when running on large numbers of variants. Consider running vep followed by filter_vep to get a smaller input set:
	
	`./vep -i input.vcf -cache -vcf -o input_vep.vcf`
	
	`./filter_vep -i input_vep.vcf -filter "Consequence is missense_variant" > input_vep_filtered.vcf`
	
	`./vep -i input_vep_filtered.vcf -cache -plugin LD`

LocalID	The LocalID plugin allows you to use variant IDs as input without making a database connection. ... Requires sqlite3. A local sqlite3 database is used to look up variant IDs; this is generated either from Ensembl's public database (very slow, but includes synonyms), or from a VEP cache file (faster, excludes synonyms). NB this plugin is NOT compatible with the ensembl-tools variant_effect_predictor.pl version of VEP.
LoFtool Loss-of-function	Add LoFtool scores to the VEP output. ... LoFtool provides a rank of genic intolerance and consequent susceptibility to disease based on the ratio of Loss-of-function (LoF) to synonymous mutations for each gene in 60,706 individuals from ExAC, adjusting for the gene de novo mutation rate and evolutionary protein conservation. The lower the LoFtool gene score percentile the most intolerant is the gene to functional variation. For more details please see (Fadista J et al. 2017), PMID:27563026. The authors would like to thank the Exome Aggregation Consortium and the groups that provided exome variant data for comparison. A full list of contributing groups can be found at http://exac.broadinstitute.org/about. The LoFtool_scores.txt file is found alongside the plugin in the VEP_plugins GitHub repo. To use another scores file, add it as a parameter i.e. `./vep -i variants.vcf --plugin LoFtool,scores_file.txt`
LOVD Leiden Open Variation Database	A VEP plugin that retrieves LOVD variation data from http://www.lovd.nl/. ... Please be aware that LOVD is a public resource of curated variants, therefore please respect this resource and avoid intensive querying of their databases using this plugin, as it will impact the availability of this resource for others.

Plugin	Description

Mastermind	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that
	uses the Mastermind Genomic Search Engine (https://www.genomenon.com/mastermind)
	to report variants that have clinical evidence cited in the medical literature.
	It is available for both GRCh37 and GRCh38. ...
	
	Running options:
	(Option 1) By default, this plugin matches the citation data with the specific mutation.
	(Option 2) It can be run with the flag '1' to return the citations for all mutations/transcripts.
	
	Output:
	The output includes three unique counts 'MMCNT1, MMCNT2, MMCNT3' and one identifier 'MMID3'
	to be used to build an URL which shows all articles from MMCNT3.
	
	'MMCNT1' is the count of Mastermind articles with cDNA matches for a specific variant;
	'MMCNT2' is the count of Mastermind articles with variants either explicitly matching at
	the cDNA level or given only at protein level;
	'MMCNT3' is the count of Mastermind articles including other DNA-level variants resulting
	in the same amino acid change;
	'MMID3' is the Mastermind variant identifier(s), as gene:key, for MMCNT3;
	
	To build the URL, substitute the 'gene:key' in the following link with the value from MMID3:
	https://mastermind.genomenon.com/detail?disease=all%20diseases&gene=gene&mutation=gene:key
	
	More information can be found at: https://www.genomenon.com/cvr/
	
	The following steps are necessary before running this plugin:
	
	Download and Registry (free):
	https://www.genomenon.com/cvr/
	
	GRCh37 VCF:
	unzip mastermind_cited_variants_reference-XXXX.XX.XX-grch37-vcf.zip
	bgzip mastermind_cited_variants_reference-XXXX.XX.XX-GRCh37-vcf
	tabix -p vcf mastermind_cited_variants_reference-XXXX.XX.XX.GRCh37-vcf.gz
	
	GRCh38 VCF:
	unzip mastermind_cited_variants_reference-XXXX.XX.XX-grch38-vcf.zip
	bgzip mastermind_cited_variants_reference-XXXX.XX.XX-GRCh38-vcf
	tabix -p vcf mastermind_cited_variants_reference-XXXX.XX.XX.GRCh38-vcf.gz
	
	The plugin can then be run as default (Option 1):
	
	`./vep -i variations.vcf --plugin Mastermind,/path/to/mastermind_cited_variants_reference-XXXX.XX.XX.GRChXX-vcf.gz`
	
	or with an option to not filter by mutations (Option 2):
	
	`./vep -i variations.vcf --plugin Mastermind,/path/to/mastermind_cited_variants_reference-XXXX.XX.XX.GRChXX-vcf.gz,1`
	
	Note: While running this plugin as default, i.e. filtering by mutation, if a variant doesn't affect
	the protein sequence, the citation data can be appended to a transcript with different consequence.
	Example
	VEP: upstream_gene_variant
	Mastermind: intronic
	VEP output: var_1l1:154173185-154173187lClENSG00000143549lENST00000368545lTranscriptlupstream_gene_variantl
	-l-l-l-l-l-lIMPACT=MODIFIER;DISTANCE=508;STRAND=-1;Mastermind_MMID3=TPM3:E62int;Mastermind_counts=1l1l1;

MaxEntScan	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that runs MaxEntScan (http://genes.mit.edu/burgelab/maxent/Xmaxentscan_scoreseq.html) to get splice site predictions. ... The plugin copies most of the code verbatim from the score5.pl and score3.pl scripts provided in the MaxEntScan download. To run the plugin you must get and unpack the archive from http://genes.mit.edu/burgelab/maxent/download/; the path to this unpacked directory is then the param you pass to the --plugin flag. The plugin executes the logic from one of the scripts depending on which splice region the variant overlaps: score5.pl : last 3 bases of exon --> first 6 bases of intron score3.pl : last 20 bases of intron --> first 3 bases of exon The plugin reports the reference, alternate and difference (REF - ALT) maximum entropy scores. If 'SWA' is specified as a command-line argument, a sliding window algorithm is applied to subsequences containing the reference and alternate alleles to identify k-mers with the highest donor and acceptor splice site scores. To assess the impact of variants, reference comparison scores are also provided. For SNVs, the comparison scores are derived from sequence in the same frame as the highest scoring k-mers containing the alternate allele. For all other variants, the comparison scores are derived from the highest scoring k-mers containing the reference allele. The difference between the reference comparison and alternate scores (SWA_REF_COMP - SWA_ALT) are also provided. If 'NCSS' is specified as a command-line argument, scores for the nearest upstream and downstream canonical splice sites are also included. By default, only scores are reported. Add 'verbose' to the list of command- line arguments to include the sequence output associated with those scores.

Plugin	Description
MPC missense deleteriousness metric	A VEP plugin that retrieves MPC scores for variants from a tabix-indexed MPC data file. ... MPC is a missense deleteriousness metric based on the analysis of genic regions depleted of missense mutations in the Exome Agggregation Consortium (ExAC) data. The MPC score is the product of work by Kaitlin Samocha (ks20@sanger.ac.uk). Publication currently in pre-print: Samocha et al bioRxiv 2017 (TBD) The MPC score file is available to download from: ftp://ftp.broadinstitute.org/pub/ExAC_release/release1/regional_missense_constraint/ The data are currently mapped to GRCh37 only. Not all transcripts are included; see README in the above directory for exclusion criteria.
MTR Missense Tolerance Ratio	A VEP plugin that retrieves Missense Tolerance Ratio (MTR) scores for variants from a tabix-indexed flat file. ... MTR scores quantify the amount of purifying selection acting specifically on missense variants in a given window of protein-coding sequence. It is estimated across a sliding window of 31 codons and uses observed standing variation data from the WES component of the Exome Aggregation Consortium Database (ExAC), version 2.0 (http://gnomad.broadinstitute.org). Please cite the MTR publication alongside the VEP if you use this resource: http://genome.cshlp.org/content/27/10/1715 The Bio::DB::HTS perl library or tabix utility must be installed in your path to use this plugin. MTR flat files can be downloaded from: ftp://mtr-viewer.mdhs.unimelb.edu.au/pub NB: Data are available for GRCh37 only
NearestExonJB	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that finds the nearest exon junction boundary to a coding sequence variant. More than one boundary may be reported if the boundaries are equidistant. ... The plugin will report the Ensembl identifier of the exon, the distance to the exon boundary, the boundary type (start or end of exon) and the total length in nucleotides of the exon. Various parameters can be altered by passing them to the plugin command: - max_range : maximum search range in bp (default: 10000) Parameters are passed e.g.: `--plugin NearestExonJB,max_range=50000`
NearestGene	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that finds the nearest gene(s) to a non-genic variant. More than one gene may be reported if the genes overlap the variant or if genes are equidistant. ... Various parameters can be altered by passing them to the plugin command: - limit : limit the number of genes returned (default: 1) - range : initial search range in bp (default: 1000) - max_range : maximum search range in bp (default: 10000) Parameters are passed e.g.: `--plugin NearestGene,limit=3,max_range=50000` This plugin requires a database connection. It cannot be run with VEP in offline mode i.e. using the --offline flag.

Plugin	Description

neXtProt	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that retrieves data for missense variants from neXtProt, which is a comprehensive human-centric discovery platform that offers integration of and navigation through protein-related data (https://www.nextprot.org/). ...
	
	Please cite the neXtProt publication alongside the VEP if you use this resource: https://doi.org/10.1093/nar/gkz995
	
	This plugin is only suitable for small sets of variants as an additional individual remote API query run for each variant.
	
	Running options:
	(Default) the data retrieved by default is the MatureProtein, NucleotidePhosphateBindingRegion, Variant, MiscellaneousRegion, TopologicalDomain and InteractingRegion.
	The plugin can also be run with other options to retrieve other data than the default.
	
	Options are passed to the plugin as key=value pairs:
	max_set : Set value to 1 to return all available protein-related data (includes the default data)
	
	return_values : The set of data to be returned.
	Use file 'neXtProt_headers.txt' to check which data (labels) are available.
	
	url : Set value to 1 to include the URL to link to the neXtProt entry.
	
	all_labels : Set value to 1 to include all labels, even if data is not available.
	
	position : Set value to 1 to include the start and end position in the protein.
	
	* note: 'max_set' and 'return_values' cannot be used simultaneously.
	
	Output:
	By default, the plugin only returns data that is available. Example (default behaviour):
	neXtProt_MatureProtein=Rho guanine nucleotide exchange factor 10
	
	The option 'all_labels' returns a consistent set of the requested fields, using "-" where values are not available. Same example as above:
	neXtProt_MatureProtein=Rho guanine nucleotide exchange factor 10;
	neXtProt_InteractingRegion=-;neXtProt_NucleotidePhosphateBindingRegion=-;neXtProt_Variant=-;
	neXtProt_MiscellaneousRegion=-;neXtProt_TopologicalDomain=-;
	
	The plugin can then be run as default:
	
	`./vep -i variations.vcf --plugin neXtProt`
	
	or to return only the data specified by the user:
	
	`./vep -i variations.vcf --plugin neXtProt,return_values='Domain&InteractingRegion'`

Plugin	Description

Phenotypes	A VEP plugin that retrieves overlapping phenotype information. ...
	
	On the first run for each new version/species/assembly will
	download a GFF-format dump to ~/.vep/Plugins/
	
	Ensembl provides phenotype annotations mapped to a number of genomic
	feature types, including genes, variants and QTLs.
	
	This plugin is best used with JSON output format; the output will be
	more verbose and include all available phenotype annotation data and
	metadata.
	
	For other output formats, only a concatenated list of phenotype
	description strings is returned.
	
	Several paramters can be set using a key=value system:
	
	dir : provide a dir path, where either to create anew the species
	specific file from the download or to look for an existing file
	
	file : provide a file path, either to create anew from the download
	or to point to an existing file
	
	exclude_sources: exclude sources of phenotype information. By default
	HGMD and COSMIC annotations are excluded. See
	http://www.ensembl.org/info/genome/variation/phenotype/sources_phenotype_documentation.html
	Separate multiple values with '&'
	
	include_sources: force include sources, as exclude_sources
	
	exclude_types : exclude types of features. By default StructuralVariation
	and SupportingStructuralVariation annotations are excluded
	due to their size. Separate multiple values with '&'.
	Valid types: Gene, Variation, QTL, StructuralVariation,
	SupportingStructuralVariation, RegulatoryFeature
	
	include_types : force include types, as exclude_types
	
	expand_right : sets cache size in bp. By default annotations 100000bp (100kb)
	downstream of the initial lookup are cached
	
	phenotype_feature : report the specific gene or variation the phenotype is
	linked to, this can be an overlapping gene or structural variation,
	and the source of the annotation (default 0)
	
	Example:
	
	`--plugin Phenotypes,file=${HOME}/phenotypes.gff.gz,include_types=Gene`
	
	`--plugin Phenotypes,dir=${HOME},include_types=Gene`

PON_P2	This plugin for Ensembl Variant Effect Predictor (VEP) computes the predictions of PON-P2 for amino acid substitutions in human proteins. PON-P2 is developed and maintained by Protein Structure and Bioinformatics Group at Lund University and is available at http://structure.bmc.lu.se/PON-P2/. ... To run this plugin, you will require a python script and its dependencies (Python, python suds). The python file can be downloaded from http://structure.bmc.lu.se/PON-P2/vep.html/ and the complete path to this file must be supplied while using this plugin.

Plugin	Description

PostGAP	A VEP plugin that retrieves data for variants from a tabix-indexed PostGAP file (1-based file). ...
	
	Please refer to the PostGAP github and wiki for more information:
	https://github.com/Ensembl/postgap
	https://github.com/Ensembl/postgap/wiki
	https://github.com/Ensembl/postgap/wiki/algorithm-pseudo-code
	
	The Bio::DB::HTS perl library or tabix utility must be installed in your path
	to use this plugin. The PostGAP data file can be downloaded from
	https://storage.googleapis.com/postgap-data.
	
	The file must be processed and indexed by tabix before use by this plugin.
	PostGAP has coordinates for both GRCh38 and GRCh37; the file must be
	processed differently according to the assembly you use.
	
	> wget https://storage.googleapis.com/postgap-data/postgap.txt.gz
	> gunzip postgap.txt.gz
	
	# GRCh38
	> (grep ^"ld_snp_rsID" postgap.txt; grep -v ^"ld_snp_rsID" postgap.txt | sort -k4,4 -k5,5n) | bgzip > postgap_GRCh38.txt.gz
	> tabix -s 4 -b 5 -e 5 -c l postgap_GRCh38.txt.gz
	
	# GRCh37
	> (grep ^"ld_snp_rsID" postgap.txt; grep -v ^"ld_snp_rsID" postgap.txt | sort -k2,2 -k3,3n) | bgzip > postgap_GRCh37.txt.gz
	> tabix -s 2 -b 3 -e 3 -c l postgap_GRCh37.txt.gz
	
	Note that in the last command we tell tabix that the header line starts with "l";
	this may change to the default of "#" in future versions of PostGAP.
	
	When running the plugin by default 'disease_efo_id', 'disease_name', 'gene_id'
	and 'score' information is returned e.g.
	
	`--plugin POSTGAP,/path/to/PostGap.gz`
	
	You may include all columns with ALL; this fetches a large amount of data per
	variant!:
	
	`--plugin POSTGAP,/path/to/PostGap.gz,ALL`
	
	You may want to select only a specific subset of additional information to be
	reported, you can do this by specifying the columns as parameters to the plugin e.g.
	
	`--plugin POSTGAP,/path/to/PostGap.gz,gwas_pmid,gwas_size`
	
	If a requested column is not found, the error message will report the
	complete list of available columns in the POSTGAP file. For a brief description
	of the available information please refer to the 'How do I use POSTGAP output?'
	section in the POSTGAP wiki.
	
	Tabix also allows the data file to be hosted on a remote server. This plugin is
	fully compatible with such a setup - simply use the URL of the remote file:
	
	`--plugin PostGAP,http://my.files.com/postgap.txt.gz`
	
	Note that gene sequences referred to in PostGAP may be out of sync with
	those in the latest release of Ensembl; this may lead to discrepancies with
	scores retrieved from other sources.

ProteinSeqs	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that prints out the reference and mutated protein sequences of any proteins found with non-synonymous mutations in the input file. ... You should supply the name of file where you want to store the reference protein sequences as the first argument, and a file to store the mutated sequences as the second argument. Note that, for simplicity, where stop codons are gained the plugin simply substitutes a '*' into the sequence and does not truncate the protein. Where a stop codon is lost any new amino acids encoded by the mutation are appended to the sequence, but the plugin does not attempt to translate until the next downstream stop codon. Also, the protein sequence resulting from each mutation is printed separately, no attempt is made to apply multiple mutations to the same protein.

Plugin	Description

ReferenceQuality	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that
	reports on the quality of the reference genome using GRC data at the location of your variants.
	More information can be found at: https://www.ncbi.nlm.nih.gov/grc/human/issues ...
	
	The following steps are necessary before running this plugin:
	
	GRCh38:
	
	Download
	ftp://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/GRCh38/MISC/annotated_clone_assembly_problems_GCF_000001405.38.gff3
	ftp://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/Issue_Mapping/GRCh38.p12_issues.gff3
	
	cat annotated_clone_assembly_problems_GCF_000001405.38.gff3 GRCh38.p12_issues.gff3 > GRCh38_quality_mergedfile.gff3
	sort -k 1,1 -k 4,4n -k 5,5n GRCh38_quality_mergedfile.gff3 > sorted_GRCh38_quality_mergedfile.gff3
	bgzip sorted_GRCh38_quality_mergedfile.gff3
	tabix -p gff sorted_GRCh38_quality_mergedfile.gff3.gz
	
	The plugin can then be run with:
	
	`./vep -i variations.vcf --plugin ReferenceQuality,sorted_GRCh38_quality_mergedfile.gff3.gz`
	
	GRCh37:
	
	Download
	ftp://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/GRCh37/MISC/annotated_clone_assembly_problems_GCF_000001405.25.gff3
	ftp://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/Issue_Mapping/GRCh37.p13_issues.gff3
	
	cat annotated_clone_assembly_problems_GCF_000001405.25.gff3 GRCh37.p13_issues.gff3 > GRCh37_quality_mergedfile.gff3
	sort -k 1,1 -k 4,4n -k 5,5n GRCh37_quality_mergedfile.gff3 > sorted_GRCh37_quality_mergedfile.gff3
	bgzip sorted_GRCh37_quality_mergedfile.gff3
	tabix -p gff sorted_GRCh37_quality_mergedfile.gff3.gz
	
	The plugin can then be run with:
	
	`./vep -i variations.vcf --plugin ReferenceQuality,sorted_GRCh37_quality_mergedfile.gff3.gz`
	
	The tabix utility must be installed in your path to use this plugin.

REVEL	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds the REVEL score for missense variants to VEP output. ... Please cite the REVEL publication alongside the VEP if you use this resource: https://www.ncbi.nlm.nih.gov/pubmed/27666373 REVEL scores can be downloaded from: https://sites.google.com/site/revelgenomics/downloads and can be tabix-processed by: cat revel_all_chromosomes.csv | tr "," "\t" > tabbed_revel.tsv sed '1s/.*/#&/' tabbed_revel.tsv > new_tabbed_revel.tsv bgzip new_tabbed_revel.tsv tabix -f -s 1 -b 2 -e 2 new_tabbed_revel.tsv.gz The tabix utility must be installed in your path to use this plugin.
SameCodon	A VEP plugin that reports existing variants that fall in the same codon.

Plugin	Description

satMutMPRA	A VEP plugin that retrieves data for variants from a tabix-indexed satMutMPRA file (1-based file).
	The saturation mutagenesis-based massively parallel reporter assays (satMutMPRA) measures variant effects on gene RNA expression for 21 regulatory elements (11 enhancers, 10 promoters). ...
	
	The 20 disease-associated regulatory elements and one ultraconserved enhancer analysed in different cell lines are the following:
	- ten promoters (of TERT, LDLR, HBB, HBG, HNF4A, MSMB, PKLR, F9, FOXE1 and GP1BB) and
	- ten enhancers (of SORT1, ZRS, BCL11A, IRF4, IRF6, MYC (2x), RET, TCF7L2 and ZFAND3) and
	- one ultraconserved enhancer (UC88).
	
	Please refer to the satMutMPRA web server and Kircher M et al. (2019) paper for more information:
	https://mpra.gs.washington.edu/satMutMPRA/
	https://www.ncbi.nlm.nih.gov/pubmed/31395865
	
	Parameters can be set using a key=value system:
	file : required - a tabix indexed file of the satMutMPRA data corresponding to desired assembly.
	
	pvalue : p-value threshold (default: 0.00001)
	
	cols : colon delimited list of data types to be returned from the satMutMPRA data
	(default: 'Value', 'P-Value', and 'Element')
	
	incl_repl : include replicates (default: off):
	- full replicate for LDLR promoter (LDLR.2) and SORT1 enhancer (SORT1.2)
	- a reversed sequence orientation for SORT1 (SORT1-flip)
	- other conditions: PKLR-48h, ZRSh-13h2, TERT-GAa, TERT-GBM, TERG-GSc
	
	The Bio::DB::HTS perl library or tabix utility must be installed in your path
	to use this plugin. The satMutMPRA data file can be downloaded from
	https://mpra.gs.washington.edu/satMutMPRA/
	
	satMutMPRA data can be downloaded for both GRCh38 and GRCh37 from the web server (https://mpra.gs.washington.edu/satMutMPRA/):
	'Download' section, select 'GRCh37' or 'GRCh38' for 'Genome release' and 'Download All Elements'.
	
	The file must be processed and indexed by tabix before use by this plugin.
	
	# GRCh38
	> (grep ^Chr GRCh38_ALL.tsv; grep -v ^Chr GRCh38_ALL.tsv | sort -k1,1 -k2,2n) | bgzip > satMutMPRA_GRCh38_ALL.gz
	> tabix -s 1 -b 2 -e 2 -c C satMutMPRA_GRCh38_ALL.gz
	
	# GRCh37
	> (grep ^Chr GRCh37_ALL.tsv; grep -v ^Chr GRCh37_ALL.tsv | sort -k1,1 -k2,2n) | bgzip > satMutMPRA_GRCh37_ALL.gz
	> tabix -s 1 -b 2 -e 2 -c C satMutMPRA_GRCh37_ALL.gz
	
	When running the plugin by default 'Value', 'P-Value', and 'Element'
	information is returned e.g.
	
	`--plugin satMutMPRA,file=/path/to/satMutMPRA_GRCh38_ALL.gz`
	
	You may include all columns with ALL; this fetches all data per variant
	(e.g. Tags, DNA, RNA, Value, P-Value, Element):
	
	`--plugin satMutMPRA,file=/path/to/satMutMPRA_GRCh38_ALL.gz,cols=ALL`
	
	You may want to select only a specific subset of information to be
	reported, you can do this by specifying the specific columns as parameters to the plugin e.g.
	
	`--plugin satMutMPRA,file=/path/to/satMutMPRA_GRCh38_ALL.gz,cols=Tags:DNA`
	
	If a requested column is not found, the error message will report the
	complete list of available columns in the satMutMPRA file. For a detailed description
	of the available information please refer to the manuscript or online web server.
	
	Tabix also allows the data file to be hosted on a remote server. This plugin is
	fully compatible with such a setup - simply use the URL of the remote file:
	
	`--plugin satMutMPRA,file=http://my.files.com/satMutMPRA.gz`
	
	Note that gene locations referred to in satMutMPRA may be out of sync with
	those in the latest release of Ensembl; this may lead to discrepancies with
	information retrieved from other sources.

SingleLetterAA	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that returns a HGVSp string with single amino acid letter codes

Plugin	Description
SpliceAI	A VEP plugin that retrieves pre-calculated annotations from SpliceAI. SpliceAI is a deep neural network, developed by Illumina, Inc that predicts splice junctions from an arbitrary pre-mRNA transcript sequence. ... Delta score of a variant, defined as the maximum of (DS_AG, DS_AL, DS_DG, DS_DL), ranges from 0 to 1 and can be interpreted as the probability of the variant being splice-altering. The author-suggested cutoffs are: 0.2 (high recall) 0.5 (recommended) 0.8 (high precision) This plugin is available for both GRCh37 and GRCh38. More information can be found at: https://pypi.org/project/spliceai/ Please cite the SpliceAI publication alongside VEP if you use this resource: https://www.ncbi.nlm.nih.gov/pubmed/30661751 Running options: (Option 1) By default, this plugin appends all scores from SpliceAI files. (Option 2) Besides the pre-calculated scores, it can also be specified a score cutoff between 0 and 1. Output: The output includes the gene symbol, delta scores (DS) and delta positions (DP) for acceptor gain (AG), acceptor loss (AL), donor gain (DG), and donor loss (DL). For tab and JSON the output contains one header 'SpliceAI_pred' with all the delta scores and positions. The format is: SYMBOLIDS_AGIDS_ALIDS_DGIDS_DLIDP_AGIDP_ALIDP_DGIDP_DL For VCF output the delta scores and positions are stored in different headers. The values are 'SpliceAI_pred_xx' being 'xx' the score/position. Example: 'SpliceAI_pred_DS_AG' is the delta score for acceptor gain. If plugin is run with option 2, the output also contains a flag: 'PASS' if delta score passes the cutoff, 'FAIL' otherwise. The following steps are necessary before running this plugin: The files with the annotations for all possible substitutions (snv), 1 base insertions and 1-4 base deletions (indel) within genes are available here: https://basespace.illumina.com/s/otSPW8hnhaZR GRCh37: tabix -p vcf spliceai_scores.raw.snv.hg37.vcf.gz tabix -p vcf spliceai_scores.raw.indel.hg37.vcf.gz GRCh38: tabix -p vcf spliceai_scores.raw.snv.hg38.vcf.gz tabix -p vcf spliceai_scores.raw.indel.hg38.vcf.gz The plugin can then be run: `./vep -i variations.vcf --plugin SpliceAI,snv=/path/to/spliceai_scores.raw.snv.hg38.vcf.gz,` indel=/path/to/spliceai_scores.raw.indel.hg38.vcf.gz `./vep -i variations.vcf --plugin SpliceAI,snv=/path/to/spliceai_scores.raw.snv.hg38.vcf.gz,` indel=/path/to/spliceai_scores.raw.indel.hg38.vcf.gz,cutoff=0.5
SpliceRegion	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that provides more granular predictions of splicing effects. ... Three additional terms may be added: # splice_donor_5th_base_variant : variant falls in the 5th base after the splice donor junction (5' end of intron) v ...EEEEEIIIIIIIII... (E = exon, I = intron, v = variant location) # splice_donor_region_variant : variant falls in region between 3rd and 6th base after splice junction (5' end of intron) vv vvv ...EEEEEIIIIIIIII... # splice_polypyrimidine_tract_variant : variant falls in polypyrimidine tract at 3' end of intron, between 17 and 3 bases from the end vvvvvvvvvvvvv ...IIIIIIIIIIIIIIIIIEEEEE...

Plugin	Description
StructuralVariantOverlap	A VEP plugin that retrieves information from overlapping structural variants. ... Parameters can be set using a key=value system: file : required - a VCF file of reference data. percentage : percentage overlap between SVs (default: 80) reciprocal : calculate reciprocal overlap, options: 0 or 1. (default: 0) (overlap is expressed as % of input SV by default) cols : colon delimited list of data types to return from the INFO fields (only AF by default) same_type : 1/0 only report SV of the same type (eg deletions for deletions, off by default) distance : the distance the ends of the overlapping SVs should be within. match_type : only report reference SV which lie within or completely surround the input SV options: within, surrounding label : annotation label that will appear in the output (default: "SV_overlap") Example- input: label=mydata, output: mydata_name=refSV,mydata_PC=80,mydata_AF=0.05 Example reference data 1000 Genomes Project: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/integrated_sv_map/ALL.wgs.mergedSV.v8.20130502.svs.genotypes.vcf.gz gnomAD:: https://storage.googleapis.com/gnomad-public/papers/2019-sv/gnomad_v2_sv.sites.vcf.gz Example: `./vep -i structvariants.vcf --plugin StructuralVariantOverlap,file=gnomad_v2_sv.sites.vcf.gz`
SubsetVCF	A VEP plugin to retrieve overlapping records from a given VCF file. Values for POS, ID, and ALT, are retrieved as well as values for any requested INFO field. Additionally, the allele number of the matching ALT is returned. ... Though similar to using '--custom', this plugin returns all ALTs for a given POS, as well as all associated INFO values. By default, only VCF records with a filter value of "PASS" are returned, however this behaviour can be changed via the 'filter' option. Parameters: name: short name added used as a prefix (required) file: path to tabix-index vcf file (required) filter: only consider variants marked as 'PASS', 1 or 0 (default, 1) fields: info fields to be returned (default, not used) '%' can delimit multiple fields '*' can be used as a wildcard Returns: _POS: POS field from VCF _REF: REF field from VCF (minimised) _ALT: ALT field from VCF (minimised) _alt_index: Index of matching variant (zero-based) _: List of requested info values
TSSDistance	A VEP plugin that calculates the distance from the transcription start site for upstream variants.

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 you to share any plugins you develop: we are happy to accept pull requests on the VEP_plugins git repository.

How it works

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

Functionality

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

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

Filtering using plugins

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

method to return true if you want to include this line, and false otherwise.
Again, please refer to the documentation in this module for more details and an example implementation of a missense filter.

Using plugins

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

For example, if your plugin is in a module called *MyPlugin.pm*, stored in *~/.vep/Plugins*, you can run it with a command line like:

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

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

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

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

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

Intergenic variants

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

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

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

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

Example commands

- Read input from **STDIN**, output to **STDOUT**

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

- Add **regulatory** region **consequences**

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

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

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

- **Filter** out **common variants** based on 1000 Genomes data

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

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

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

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

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

- Connect to **Ensembl Genomes** db server for *Arabidopsis thaliana*

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

- Load config from **ini file**, run in **quiet mode**

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

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

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

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

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

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

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

- Right align variants before consequence calculation. For more information, see here.

```
./vep --cache -i variants.vcf --shift_3prime 1
```

gnomAD and ExAC

gnomAD exome frequency data is included in VEP's cache files from release 90, replacing ExAC; use --af_gnomad to enable using this data. VEP can also retrieve frequency data from the gnomAD genomes set or ExAC via VEP's custom annotation functionality.

1. VEP requires Bio::DB::HTS to read data from tabix-indexed VCFs - see installation instructions
2. Ensembl's FTP site hosts abridged VCF files for gnomAD and ExAC, additionally remapped to GRCh38 using CrossMap. It is possible for VEP to read these files directly from their remote location, though for optimal performance the VCF and index should be downloaded to a local file system.
   - **GRCh38**
     - gnomAD genomes (r2.1, remapped with CrossMap): [VCFs and tabix indexes]
     - gnomAD exomes (r2.1, remapped with CrossMap): [VCFs and tabix indexes]
     - ExAC (v0.3, remapped using CrossMap): [VCF] [tabix index]
   - **GRCh37**
     - gnomAD genomes (r2.1): [VCF and tabix indexes]
     - gnomAD exomes (r2.1): [VCF and tabix indexes]
     - ExAC (v0.3): [VCF] [tabix index]
3. Run VEP with the following command (using the GRCh38 input example) to get locations and continental-level allele frequencies:

```
./vep -i examples/homo_sapiens_GRCh38.vcf --cache \
--custom gnomad.genomes.r2.0.1.sites.GRCh38.noVEP.vcf.gz,gnomADg,vcf,exact,0,AF_AFR,AF_AMR,AF_ASJ,AF_EAS,AF_FIN,AF_NFE,AF_OTH
```

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

```
## gnomADg : gnomad.genomes.r2.0.1.sites.GRCh38.noVEP.vcf.gz (exact)
## gnomADg_AFR_AF : AFR_AF field from gnomad.genomes.r2.0.1.sites.GRCh38.noVEP.vcf.gz
## gnomADg_AMR_AF : AMR_AF field from gnomad.genomes.r2.0.1.sites.GRCh38.noVEP.vcf.gz
...
```

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

Conservation scores

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

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

Example conservation score files:

Human (GRCh38)

- phastCons 7-way
- phastCons 20-way
- phastCons 100-way
- phyloP 7-way
- phyloP 20-way
- phyloP 100-way

Human (GRCh37)

- GERP
- phastCons 46-way
- phastCons 100-way
- phyloP 46-way
- phyloP 100-way

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

---

dbNSFP

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

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

```
unzip dbNSFP4.0b2a.zip
head -n1 dbNSFP4.0b2a_variant.chr1 > dbNSFP4.0b2a.txt
cat dbNSFP4.0b2a_variant.chr* | grep -v "#" >> dbNSFP4.0b2a.txt
rm dbNSFP4.0b2a_variant.chr*
bgzip dbNSFP4.0b2a.txt
tabix -s 1 -b 2 -e 2 dbNSFP4.0b2a.txt.gz
```

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

```
./vep --cache --force --plugin dbNSFP,dbNSFP4.0b2a.txt.gz
2014-04-04 11:27:05 - Read existing cache info
2014-04-04 11:27:05 - Auto-detected FASTA file in cache directory
2014-04-04 11:27:05 - Checking/creating FASTA index
2014-04-04 11:27:05 - Failed to instantiate plugin dbNSFP: ERROR: No columns selected to fetch. 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,
...
```

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

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

```
./vep --cache --force --plugin dbNSFP,dbNSFP4.0b2a.txt.gz,LRT_score,FATHM_score,MutationTaster_score
```

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

Structural Variants

VEP can be used to annotate structural variants with their predicted effect on other genomic features. Input data should be supplied as VCF.

Prediction process

- The INFO keys 'END' or 'SVLEN' are present, the proportion of any overlapping feature covered by the variant is calculated
- If the SVTYPE or ALT is 'DEL', the variant tested for feature ablation/ truncation
- If the SVTYPE or ALT is 'DUP', the variant tested for feature amplification
- If the SVTYPE or ALT is 'INS' or 'DUP', the variant tested for feature elongatation
- SVTYPE is used in preference to ALT to derive the variant type of an SV with 'CN*' alleles

Reported overlaps

- VEP calculates the length and proportion of each genomic feature overlapped by a structural variant
- Use the --overlaps option to enable this when using VCF or tab format. (This is reported by default in standard VEP and JSON format.)

- The keys bp_overlap and percentage_overlap are used in JSON format and OverlapBP and OverlapPC in other formats.

Changing memory requirements

- By default, VEP does not annotate variants larger than 10M. If you are using the command line tool, you can use the --max_sv_size option to modify this.
- By default, variants are analysed in batches of 5000. Using the --buffer_size option to reduce this can reduce memory requirements, especially if your data is sparse. A smaller buffer size is essential when annotating structural variants with regulatory data.

Citations and VEP users

VEP is used by many organisations and projects:

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

Other citations and use cases:

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

Variant Effect Predictor — Other information

Getting VEP to run faster

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

1. Make sure you have the latest version of VEP and the Ensembl API. We regularly introduce optimisations, alongside the new features and bug fixes of a typical new release.
2. Download a cache file for your species. If you are using --database, you should consider using --cache or --offline instead. Any time VEP has to access data from the database (even if you have a local copy), it will be slower than accessing data in the cache on your local file system.

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

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

   Consider carefully whether you need to use these flags in your analysis.
3. If you use --check_existing or any flags that invoke it (e.g. --af, --af_1kg, --filter_common, --everything), tabix-convert your cache file. Checking for known variants using a converted cache is >100% faster than using the default format.
4. Download a FASTA file (and use the flag --fasta) if you use --hgvs or --check_ref. Again, this will prevent VEP accessing the database unnecessarily (in this case to retrieve genomic sequence).
5. Using forking enables VEP to run multiple parallel "threads", with each thread processing a subset of your input. Most modern computers have more than one processor core, so running VEP with forking enabled can give huge speed increases (3-4x faster in most cases). Even computers with a single core will see speed benefits due to overheads associated with using object-oriented code in Perl.

   To use forking, you must choose a number of forks to use with the --fork flag. We recommend using 4 forks:

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

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

   When writing plugins be aware that while the VEP code attempts to preserve the state of any plugin-specific cached data between separate forks, there may be situations where data is lost. If you find this is the case, you should disable forking in the new() method of your plugin by deleting the "fork" key from the $config hash.
6. Make sure your cache and FASTA files are stored on the fastest file system or disk you have available. If you have a lot of memory in your machine, you can even pre-copy the files to memory using tmpfs.
7. Consider if you need to generate HGVS notations (--hgvs); this is a complex annotation step that can add ~50-80% to your runtime. Note also that --hgvs is switched on by --everything.
8. Install the Set::IntervalTree Perl package. This package speeds up VEP's internals by changing how overlaps between variants and transcript components are calculated.
9. Install the Ensembl::XS package. This contains compiled versions of certain key subroutines used in VEP that will run faster than the default native Perl equivalents. Using this should improve runtime by 5-10%.
10. Add the --no_stats flag. Calculating summary statistics increases VEP runtime, so can be switched off if not required
11. VEP is optimised to run on input files that are sorted in chromosomal order. Unsorted files will still work, albeit more slowly.
12. For very large files (for example those from whole-genome sequencing), VEP process can be easily parallelised by dividing your file into chunks (e.g. by chromosome). VEP will also work with tabix-indexed, bgzipped VCF files, and so the tabix utility could be used to divide the input file:

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

Species with multiple assemblies

Ensembl currently supports the two latest human assembly versions. We provide a VEP cache using the latest software version (103) for both GRCh37 and GRCh38.

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

Once you have installed the relevant cache and FASTA file, you are then able to use VEP as normal. If you are using GRCh37 and require database access in addition to the cache (for example, to look up variant identifiers using --format id, see cache limitations), you will be warned 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
```

If you have data you wish to map to a new assembly, you can use the Ensembl assembly converter tool - if you've downloaded VEP, then you have it already! The tool is found in the ensembl-tools/scripts/assembly_converter folder. There is also an online version of the tool available. Both UCSC (liftOver) and NCBI (Remap) also provide tools for converting data between assemblies.

Summarising annotation

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

A number of options are provided to reduce the amount of output produced if this depth of annotation is not required.

Example

Input data (VCF - input.vcf)

```
##fileformat=VCFv4.2
#CHROM  POS  ID  REF  ALT
1  230710048  rs699  A  G
1  230710514  var_2  A  G,T
```

Example of VEP command and output (no "pick" option):

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

#Uploaded_variation  Location  Allele  Gene  Feature  Feature_type  Consequence  cDNA_position  CDS_position  Protein_position  Amino_acids
rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018  803  268  M/T  aTg/aCg  -  IMPACT=MODERA
rs699  1:230710048  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MODIF
var_2  1:230710514  G  ENSG00000135744  ENST00000366667  Transcript  synonymous_variant  552  337  113  L  Ttg/Ctg  -  IMPACT=LOW;S
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552  337  113  L/M  Ttg/Atg  -  IMPACT=MODERA
var_2  1:230710514  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MODIF
var_2  1:230710514  T  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MODIF
```

Options

- **--pick**

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

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

  example of VEP command and output, with the "**--pick**" option.

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

rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018  803  268  M/T  aTg/aCg  -  IMPACT=MODERATE;
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552  337  113  L/M  Ttg/Atg  -  IMPACT=MODERATE;
```

- **--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.

  example of VEP command and output, with the "**--pick_allele**" option.

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

rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018  803  268  M/T  aTg/aCg  -  IMPACT=MODERATE
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552  337  113  L/M  Ttg/Atg  -  IMPACT=MODERATE
var_2  1:230710514  G  ENSG00000135744  ENST00000366667  Transcript  synonymous_variant  552  337  113  L  Ttg/Ctg  -  IMPACT=LOW;STRA
```

- **--per_gene**

  As --pick, but chooses one annotation block per gene that the input variant overlaps.

  example of VEP command and output, with the "**--per_gene**" option.

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

rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018  803  268  M/T  aTg/aCg  -  IMPACT=MO
rs699  1:230710048  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MO
var_2  1:230710514  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MO
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552  337  113  L/M  Ttg/Atg  -  IMPACT=MO
```

- **--pick_allele_gene**

  As above, but chooses one consequence block per variant allele and gene combination.

  example of VEP command and output, with the "**--pick_allele_gene**" option.

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

rs699  1:230710048  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MO
rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018  803  268  M/T  aTg/aCg  -  IMPACT=MO
var_2  1:230710514  T  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MO
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552  337  113  L/M  Ttg/Atg  -  IMPACT=MO
var_2  1:230710514  G  ENSG00000135744  ENST00000366667  Transcript  synonymous_variant  552  337  113  L  Ttg/Ctg  -  IMPACT=LO
var_2  1:230710514  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -  -  -  -  -  -  IMPACT=MO
```

- **--flag_pick**

  Instead of choosing one block and removing the others, this option adds a flag "PICK=1" to picked annotation block, allowing you to easily filter on this later using VEP's filtering tool.

- **--flag_pick_allele**

  As above, but flags one block per allele.

- **--flag_pick_allele_gene**

  As above, but flags one block per allele and gene combination.

- **--most_severe**

  This flag reports only the consequence type of the block with the highest rank, according to this table.

  example of VEP command and output, with the "**--most_severe**" option.

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

rs699  1:230710048  -  -  -  -  missense_variant  -  -  -  -  -  -  -
var_2  1:230710514  -  -  -  -  missense_variant  -  -  -  -  -  -  -
```

- **--summary**

  This flag reports only a comma-separated list of the consequence types predicted for this variant.

  example of VEP command and output, with the "**--summary**" option.

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

rs699  1:230710048  -  -  -  -  missense_variant,downstream_gene_variant                    -  -  -  -  -  -  -
var_2  1:230710514  -  -  -  -  missense_variant,synonymous_variant,downstream_gene_variant  -  -  -  -  -  -  -
```

HGVS notations

Output

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

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

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

Input

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

By default VEP uses Ensembl transcripts as the 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. Such notations must include the version number of the transcript e.g.

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

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

RefSeq transcripts

If you prefer to exclude predicted RefSeq transcripts (those with identifiers beginning with "XM_" or "XR_") use --exclude_predicted.

Identifiers and other data

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

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

Differences to the reference genome

RefSeq transcript sequences may differ from the genome sequence to which they are aligned. Ensembl's API (and hence VEP) constructs transcript models using the genomic reference sequence. These differences are accounted for using BAM-edited transcript models. in human cache files from release 90 onwards. Prior to release 90 and in non-human species differences between the RefSeq sequence and the genomic sequence are not accounted for, so some annotations produced by VEP on these transcripts may be inaccurate. Most differences occur in non-coding regions, typically in UTRs at either end of transcripts or in the addition of a poly-A tail, causing minimal impact on annotation.

For human VEP cache files, each RefSeq transcript is annotated with the REFSEQ_MATCH flag indicating whether and how the RefSeq model differs from the underlying genome.

Correcting transcript models with BAM files

NCBI have released BAM files that contain alignments of RefSeq transcripts to the genome. From release 90 onwards, these alignments have been incorporated and used to correct the transcript models in the human RefSeq and merged cache files.

VEP's cache building process uses the sequence and alignment in the BAM to correct the RefSeq model. If the corrected model does not match the original RefSeq sequence in the BAM, the corrected model is discarded. The success or failure of the BAM edit is recorded in the BAM_EDIT field of the VEP output. Failed edits are extremely rare (< 0.01% of transcripts), but any VEP annotations produced on transcripts with a failed edit status should be interpreted with extreme caution.

Using BAM-edited transcripts causes VEP to change how alleles are interpreted from input variants. Input variants are typically encoded in VCFs that are called using the reference genome. This means that the alternate (ALT) allele as given in the VCF may correspond to the reference allele as found in the corrected RefSeq transcript model. VEP will account for this, using the corrected reference allele (by enabling --use_transcript_ref) when calculating consequences, and the GIVEN_REF and USED_REF fields in the VEP output indicate any change made. If the reference allele derived from the transcript matches any given alternate (ALT) allele, then no consequence data will be produced for this allele as it will be considered non-variant. Note that this process may also clash with any interpretation from using --check_ref, so it is recommended to avoid using this flag.

To override the behaviour of --use_transcript_ref and force VEP to use your input reference allele instead of the one derived from the transcript, you may use --use_given_ref.

VEP can also side-load BAM files at runtime to correct transcript models on-the-fly; this allows corrections to be applied for other species, where alignments are available, or when using RefSeq GFF files, rather than the cache.

```
./vep --cache --refseq -i variants.vcf --species mus_musculus --bam GCF_000001635.26_GRCm38.p6_knownrefseq_alns.bam
```

BAM files are available from NCBI:

- Human GRCh38.p13
- Human GRCh37.p13

Existing or colocated variants

Use the --check_existing flag to identify known variants colocated with input variant. VEP's known variant cache is derived from Ensembl's variation database and contains variants from dbSNP and other sources.

VEP by default uses a normalisation-based allele matching algorithm to identify known variants that match input variants. Since both input and known variants may have multiple alternate (ALT) or variant alleles, each pair of reference (REF) and ALT alleles are normalised and compared independently to arrive at potential matches. VCF permits multiple allele types to be encoded on the same line, while dbSNP assigns separate rsID identifiers to different allele types at the same locus. This means different alleles from the same input variant may be assigned different known variant identifiers.

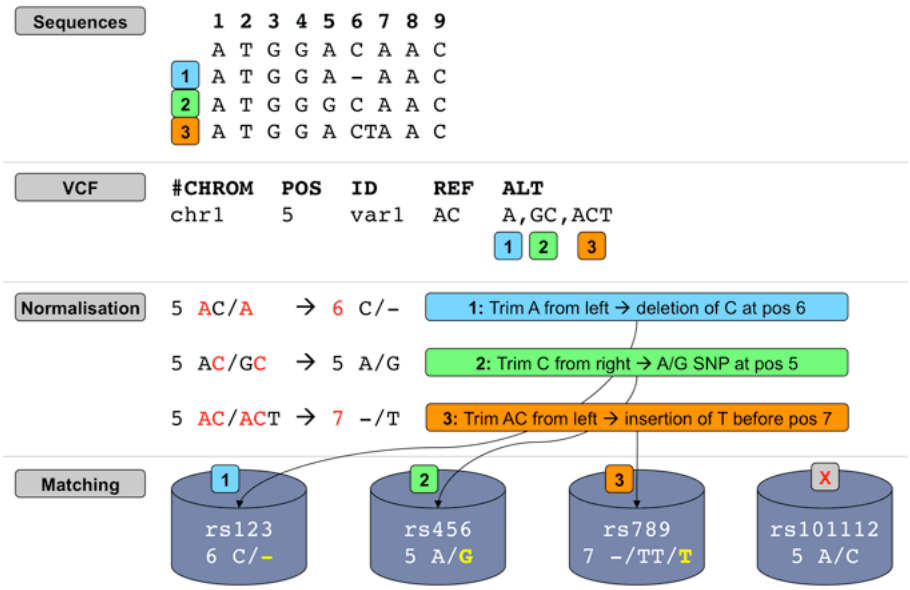


Illustration of VEP's allele matching algorithm resolving one VCF line with multiple ALTs to three different variant types and coordinates

Note that allele matching occurs independently of any allele transformations carried out by --minimal; VEP will match to the same identifiers and frequency data regardless of whether the flag is used.

For some data sources (COSMIC, HGMD), Ensembl is not licensed to redistribute allele-specific data, so VEP will report the existence of co-located variants with unknown alleles **without** carrying out allele matching. To disable this behaviour and exclude these variants, use the --exclude_null_alleles flag.

To disable allele matching completely and compare variant locations only, use --no_check_alleles.

Frequency data

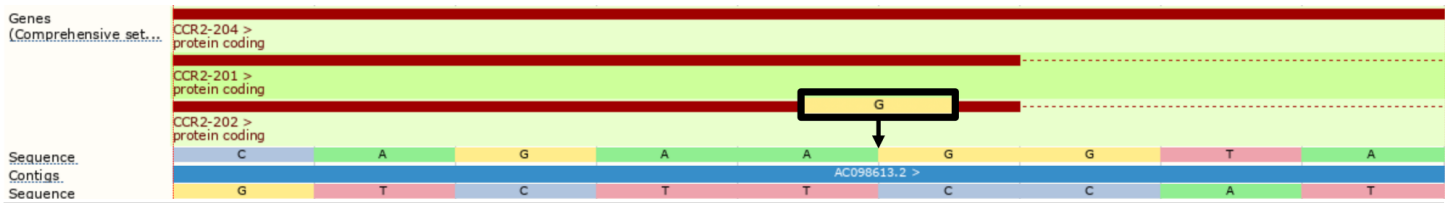
In addition to identifying known variants, VEP also reports allele frequencies for input alleles from major genotyping projects (1000 genomes, ESP and gnomAD). VEP's cache currently contains only frequency data for alleles that have been submitted to dbSNP or are imported via another source into the Ensembl variation database. This means that until gnomAD's full data set is submitted to dbSNP and incorporated into Ensembl, the frequency for some alleles may be missing from VEP's cache data.

To access the full gnomAD data set, it is possible to use VEP's custom annotation feature to retrieve the frequency data directly from the gnomAD VCF files; see instructions here.

Normalising Consequences

Insertions and deletions in repetitive sequences can be often described at different equivalent locations and may therefore be assigned different consequence predictions. VEP can optionally convert variant alleles to their most 3' representation before consequence calculation.

In the example below, we insert a G at the start of the repeated region. Without the --shift_3prime flag, VEP will calculate consequences at the input position and report the variant as a frameshift, and recognising that the variant lies within 2 bases of a splice site, as splice_region_variant.




```
./vep --cache -id '3 46358467 . A AG'
```

```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence     cDNA_position   CDS_position    Protein_pos
3_46358468_-/G  3:46358467-46358468     G       ENSG00000121807 ENST00000292301 Transcript      frameshift_variant,splice_region_variant
       IMPACT=HIGH;STRAND=1
...
```

However, with --shift_3prime switched on, VEP will right align all insertions and deletions within repeated regions, shifting the inserted G two positions to the right before consequence calculation, providing the splice_donor_variant consequence instead.

```
./vep --cache -id '3 46358467 . A AG' --shift_3prime 1
```

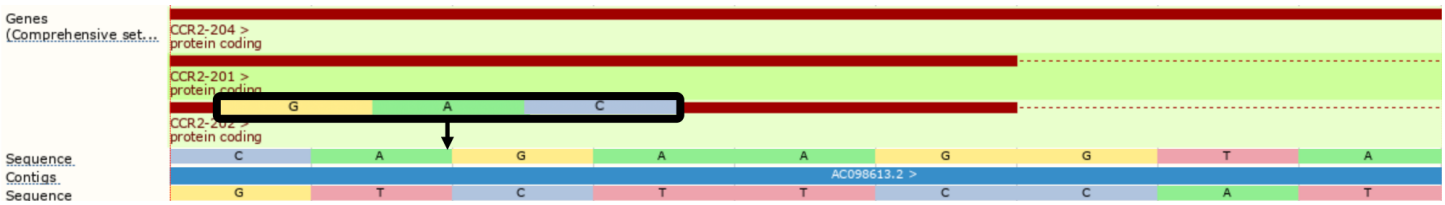
```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence     cDNA_position   CDS_position    Protein_pos
3_46358468_-/G  3:46358467-46358468     G       ENSG00000121807 ENST00000292301 Transcript      splice_donor_variant    -       -       -
...
```

Using --shift_genomic will also update the location field. However, --shift_genomic will also shift intergenic variants, which can lead to a reduction in performance.

```
./vep --cache -id '3 46358467 . A AG' --shift_genomic 1
```

```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence     cDNA_position   CDS_position    Protein_pos
3_46358468_-/G  3:46358469-46358470     G       ENSG00000121807 ENST00000292301 Transcript      splice_donor_variant    -       -       -
...
```

When shifting, insertions or deletions of length 2 or more can lead to alterations in the reported alternate allele. For example, an insertion of GAC that can be shifted 2 bases in the 3' direction will alter the alternate allele to CGA.




```
./vep --cache -id '3 46358464 . A AGAC' --shift_3prime 1
```

```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence     cDNA_position   CDS_position    Protein_pos
3_46358465_-/GAC        3:46358464-46358465     CGA     ENSG00000121807 ENST00000292301 Transcript      inframe_insertion,splice_region_var
...
```

```
./vep --cache -id '3 46358464 . A AGAC' --shift_3prime 0
```

```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence     cDNA_position   CDS_position    Protein_pos
3_46358465_-/GAC        3:46358464-46358465     GAC     ENSG00000121807 ENST00000292301 Transcript      inframe_insertion       1422-1423
```

Variant Effect Predictor FAQ

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

General questions

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

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 don't I see any co-located variants when using species X?

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

Q: Why do I see multiple known variants mapped to my input variant?

VEP compares you input to known variants from the Ensembl variation database. In some cases one input variant can match multiple known variants:

- Germline variants from dbSNP and somatic mutations from COSMIC may be found at the same locus
- Some sources, e.g. HGMD, do not provide public access to allele-specific data, so an HGMD variant with unknown alleles may colocate with one from dbSNP with known alleles
- Multiple alternate alleles from your input may match different variants as they are described in dbSNP

See here for a full discussion.

Q: VEP is not assigning a frequency to my input variant - why?

VEP's cache contains frequency data only for variants and alleles imported into Ensembl's variation database. See here for a full discussion.

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

While it would be convenient to have a simple, one word answer to the question "What is the consequence of this variant?", in reality biology is not this simple! Many genes have more than one transcript, so VEP provides a prediction for each transcript that a variant overlaps. VEP has options to help select results according to your requirements; 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.

Q: How do I reduce VEP's memory requirement?

There are a number of ways to do this-

1. Ensure your input file is sorted by location. This can greatly reduce memory requirements and runtime
2. Consider reducing the buffer size. This reduces the number of variants annotated together in a batch and can be modified in both command line and web interfaces. Reducing buffer size may increase run time.
3. Ensure you are only using the options you need, rather than --everything. Some data-rich options, such as regulatory annotation have an impact on memory use

Web VEP questions

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

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 command line VEP?

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.

Command line VEP questions

Q: How can I make VEP run faster?

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

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

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

```
substr outside of string at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/EnsEMBL/Variation/Utils/Sequence.pm line 511.
Use of uninitialized value $ref_allele in string eq at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/EnsEMBL/Variation/Utils/Seque
Use of uninitialized value in concatenation (.) or string at /nfs/users/nfs_w/wm2/Perl/ensembl-variation/modules/Bio/EnsEMBL/Variation/Utils
```

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 1.6.924 for you.

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

Q: Why do I see the following warning?

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

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

```
./vep -i input.vcf -cache -synonyms ~/.vep/homo_sapiens/103_GRCh38/chr_synonyms.txt
```

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

Q: Can I get gnomAD or ExAC allele frequencies in VEP?

Yes, see this guide.

Q: Why do I see the following error?

```
Could not connect to database homo_sapiens_core_63_37 as user anonymous using [DBI:mysql:database=homo_sapiens_core_63_37;host=ensembldb.ens
Unknown MySQL server host 'ensembldb.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;host=ensembl
Unknown MySQL server host 'ensembldb.ensembl.org' (2)
```

By default VEP is configured to connect to the public MySQL server at ensembldb.ensembl.org. Occasionally the server may break connection with your process, 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 VEP 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?

The Ensembl Variation database and the human VEP cache file contain precalculated SIFT and PolyPhen-2 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).