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.

Quick start

1. Download

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

2. Install

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

3. Test

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

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

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

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

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

Documentation contents

Download documentation in PDF format

- HGVS notations
- RefSeq transcripts
- Colocated variants
- Normalising consequences

FAQ

- General questions
- Web VEP questions
- Command line VEP questions

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 v114 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 v114 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_114_ailMel1.tar.gz
2 : anas_platyrhynchos_vep_114_BGI_duck_1.0.tar.gz
3 : anolis_carolinensis_vep_114_AnoCar2.0.tar.gz
...
42 : homo_sapiens_vep_114_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 https://ftp.ensembl.org/pub/release-
114/variation/vep/homo_sapiens_vep_114_GRCh38.tar.gz
 - unpacking homo_sapiens_vep_114_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 v114.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_114_38 on ensembldb.ensembl.org
## Using cache in /homes/user/.vep/homo_sapiens/114_GRCh38
## Using API version 114, DB version 114
## 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,SYMBOL,CANONICAL,SIFT -o STDOUT | \
./filter_vep --filter "CANONICAL is YES and SIFT is deleterious"

...

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

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

For species with an Ensembl database of variants, VEP can be configured to 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;AMR_AF=0.2003;EAS_AF=0.0456;EUR_AF=0.3211;SAS_AF=0.1401
```

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/114
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/114.zip
unzip 114.zip
cd ensembl-vep-release-114/
```

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 114 ***(October 2024)***

- Support for https protocol when downloading FTP files and adding GitHub Token to increase rate limit in VEP install script.
- Plugin support added to REST for:
  - Paralogues
- Plugin data version updated:
  - dbNSFP (from 4.7c to 4.9c)
  - LOEUF (from gnomAD v2.1.1 to gnomAD v4.1)
- Plugin deprecated:
  - DisGeNET
  - Mastermind (Only from REST)

Previous version history - from version 88:

New in version 113 ***(October 2024)***

> - gnomAD frequency data updated to v4.1 for both genomes and exomes.
> - Support for GENCODE primary transcript set added. See, --gencode_primary and --flag_gencode_primary.
> - Support added for --mane, --mane_select, and --canonical when GFF/GTF file used as annotation source.
> - Nextflow VEP now suppots other input data formats besides VCF. For supported formats see - Data formats.
> - Plugin support added to REST and Web for:
>   - RiboseqORFs
>   - REVEL
>   - ClinPred
> - Plugin support added to Web for:
>   - Paralogues
> - Plugin support added to REST for:
>   - LOEUF
> - Plugin data version updated for CADD (v1.6 to v1.7) and dbNSFP (4.5c to 4.7c).

New in version 112 ***(May 2024)***

- Enhanced Structural Variant Support:
  - Added support for CNV:TR
  - Enabled the use of chromosome synonyms in breakends
  - Report consequences for each breakend and enable the input of single breakends
- New plugins (supported on CLI, Web and REST):
  - AlphaMissense - annotates missense variants with the pre-computed AlphaMissense pathogenicity scores. AlphaMissense is a deep learning model developed by Google DeepMind that predicts the pathogenicity of single nucleotide missense variants.
- New plugins (supported on CLI and Web):
  - RiboseqORFs - uses a standardized catalog of human Ribo-seq ORFs to re-calculate consequences for variants located in these translated regions
- New plugins (supported on CLI):
  - Paralogues - fetches variants overlapping the genomic coordinates of amino acids aligned between paralogue proteins
  - AVADA - Automatic VAriant evidence DAtabase is a novel machine learning tool that uses natural language processing to automatically identify pathogenic genetic variant evidence in full-text primary literature about monogenic disease and convert it to genomic coordinates
  - GeneBe - A plugin kindly contributed by the GeneBe team, it retrieves automatic ACMG variant classification data from https://genebe.net/
  - PhenotypeOrthologous A VEP plugin that retrieves phenotype information associated with orthologous genes from model organisms
- Plugin support added to REST and Web for:
  - CADD_SV
  - CADD scores for *Sus scrofa*
  - Dosage Sensitivity
  - Enformer

New in version 111 ***(January 2024)***

- New option --individual_zyg returns a single list of individuals and their zygosity (instead of a separate line of output for each individual and variant combination like in --individual)
- Custom annotation has been improved with the following options:
  - num_records to limit the number of matching records (50 by default)
  - summary_stats to calculate summary statistics (min, mean, max, count, sum) using annotation scores (not used by default)
- New plugin (supported on CLI, REST and web):
  - OpenTargets - adds locus-to-gene (L2G) scores to predict causal genes at GWAS loci from Open Targets Genetics
- New plugin (supported on CLI and REST):
  - Enformer - adds pre-calculated predictions of variant impact on gene expression
- New plugins (supported on CLI):
  - BayesDel - adds a deleteriousness meta-score combining multiple deleteriousness predictors
  - DeNovo - identifies de novo variants in a VCF file. This plugin requires a pedigree (.ped) file
  - SpliceVault - predicts exon-skipping events and activated cryptic splice sites based on the most common mis-splicing events around a splice site
  - DosageSensitivity - annotates the likelihood of a gene being haploinsufficient or triplosensitive
  - VARITY - adds pre-calculated pathogenicity scores of rare human missense variants

New in version 110 ***(July 2023)***

- New plugins (supported on CLI):
  - TranscriptAnnotator - a VEP plugin that annotates variant-transcript pairs

- New Plugins (supported on CLI, REST and web):
  - Geno2MP - adds information from Geno2MP, a web-accessible database of rare variant genotypes linked to phenotypic information
  - MaveDB - adds information from MaveDB, a database that holds experimentally determined measures of variant effect

New in version 109 ***(February 2023)***

- VEP Docker image now includes all VEP plugins
- New plugin (supported on CLI):
  - GWAS - reports genome-wide association study data from GWAS catalog
- Plugins now available in REST and web:
  - UTRAnnotator - annotates the effect of 5' UTR variant especially for variant creating/disrupting upstream ORFs
- Plugins now available in REST:
  - NMD - predicts if a variant allows transcript to escape nonsense-mediated mRNA decay based on certain rules
- Plugin LOEUF replaces Loftool in the web with more recent 'loss-of-function' score for variants
- Deprecated Plugins:
  - miRNA - this plugin was fully deprecated in favour of --mirna flag (in web and REST)
  - ExAC - this plugin was deprecated given that VEP cache includes ExAC data as part of gnomAD
- SIFT version has been updated from 5.2.2 to 6.2.1 (except for human GRCh37)
- PolyPhen-2 version has been updated from 2.2.2 to 2.2.3 (except for human GRCh37)

New in version 108 ***(October 2022)***

- New plugin (supported on CLI, REST, and web):
  - mutfunc - predicts destabilization of protein structure, interaction and others features by a variant (GRCh38 only)
- Plugin feature extension:
  - IntAct - 4 new species are now supported - rat, chicken (red jungle fowl), yeast, and arabidopsis

New in version 107 ***(July 2022)***

- New plugin (supported on CLI, REST, and web):
  - EVE - annotates human variants using EVA classification method based solely on evolutionary sequences (GRCh38 only)
- Plugins now available in REST and web (already available in CLI):
  - GO - retrieves Gene Ontology terms associated with transcripts/translations
  - IntAct - annotates human variants which fall in interaction sites, as described in the IntAct database
- Plugins now available in web (already available in CLI):
  - NMD - predicts if a stop_gained variant allows transcript to escape nonsense-mediated mRNA decay based on certain rules
- Readthrough transcripts are now removed from cache
- Transcripts of biotype 'artifact' which are artifactual duplication are now removed from cache and not accessible using database
- gnomaAD allele frequencies are now available for exomes and genomes separately through —af_gnomade and —af_gnomadg options respectively. The —af_gnomad option have same function as --af_gnomade.

New in version 106 ***(April 2022)***

- New plugins for command line use:
  - IntAct - annotates human variants which fall in interaction sites, as described in the IntAct database
  - CAPICE - integrates scored from a machine-learning-based method for prioritizing pathogenic variants (GRCh37 only)
- Nextflow pipeline:
  - A new configurable pipeline is available to run Ensembl VEP efficiently on large scale VCF

New in version 105 ***(December 2021)***

- 3 new Sequence Ontology terms are reported for more detailed splice consequence annotation
  - splice_donor_5th_base_variant (SO:0001787)
  - splice_donor_region_variant (SO:0002170)
  - splice_polypyrimidine_tract_variant (SO:0002169)
- New plugins
  - ClinPred - adds pre-calculated scores from ClinPred which helps identify disease-relevant missense variants
  - NMD - predicts whether a stop-gained variant will allow a transcript to escape nonsense-mediated decay
- Condel scores are no longer available via the VEP web interface as they have not been updated since 2014 and newer scores like CADD and REVEL are available

New in version 104 ***(May 2021)***

- Human GRCh37 cache files now include dbSNP 154!
- --var_synonyms output structure has been altered when used with --json
- VEP Plugins:
  - dbNSFP - now supports matching by peptides
  - SpliceAI - now compares gene symbols to improve score accuracy

New in version 103 ***(February 2021)***

- **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 alongside MANE Select
  - --mane_select : Returns MANE Select annotations

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.
  - 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 ***(August 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 ***(April 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.

New in version 99 *(January 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 *(September 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 (version <=4.050)
  - 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 is available in the following platforms:

- Linux (e.g., Ubuntu, Debian, Mint)
- macOS
- Windows (requires a more involved installation process)

VEP is also available as Docker and Singularity images, allowing to skip the complex installation steps.

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 flags below. Options can also be set by exporting **environment variables** prefixed with `VEP_` before running the installer (for instance, `export VEP_NO_HTSLIB=1` and `export VEP_DIR_PLUGINS="/plugins"`).

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 114). 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! Use `all` to install data for all available species.
--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 macOS 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. macOS), 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 macOS

Installing VEP on macOS 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 macOS 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 endeavouXcoder 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** (version <=4.050)

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 macOS 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@4.050
```

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 LWP List::MoreUtils

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-114
```

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 running applications in virtualised *containers*. The VEP Docker image is available from DockerHub:

After installing Docker, download the VEP Docker image:

```
docker pull ensemblorg/ensembl-vep
```

To download cache files and other data with VEP Docker, we recommend mounting a directory from your local (host) machine to folder **`/data`** from the Docker image. For instance:

```
mkdir $HOME/vep_data
docker run -t -i -v $HOME/vep_data:/data ensemblorg/ensembl-vep
```

In the example above, data in **`$HOME/vep_data`** will be accessible by both the local machine and VEP Docker. The Ensembl VEP API, plugins and their dependencies (e.g. Perl APIs, Bio::DB::HTS, htslib, ...) are already installed in the image.

Cache and FASTA files installation

You can run the INSTALL.pl script to install the cache and FASTA files:

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

- You will be asked to install cache data. Type the comma-separated numbers for the species/assembly of interest and press `enter`. Your data will download and unpack; this may take a while.
- If you wish to retrieve HGVS annotations, please download the FASTA files for your species. To do this, at the next prompt type `0` and press `enter`.

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:/data ensemblorg/ensembl-vep INSTALL.pl -a cf -s homo_sapiens -y GRCh38
```

The installer downloads VEP data to the mounted directory (e.g., **`$HOME/vep_data`**). The downloaded data will be automatically detected as long as its folder is mounted when running VEP:

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

Running VEP with data from local folder

Here is an example on running VEP with data from folder **`$HOME/vep_data`** in the local machine (provided that the cache has been downloaded to that folder):

```
docker run -v $HOME/vep_data:/data ensemblorg/ensembl-vep \
  vep --cache --offline --format vcf --vcf --force_overwrite \
      --input_file input/my_input.vcf \
      --output_file output/my_output.vcf \
      --custom file=custom/my_extra_data.bed,short_name=BED_DATA,format=bed,type=exact,coords=1 \
      --plugin NMD
```

Please avoid using absolute paths to data as the paths inside the container differ from your local machine.

Update from a previous version

1. Update your Docker 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:/data ensemblorg/ensembl-vep INSTALL.pl -a c

# Or install the cache manually
cd $HOME/vep_data
curl -O https://ftp.ensembl.org/pub/release-
114/variation/vep/homo_sapiens_vep_114_GRCh38.tar.gz
tar xzf homo_sapiens_vep_114_GRCh38.tar.gz
```

Singularity

Due to root requirements for the Docker daemon, using the Docker container for VEP is not always possible to HPC users. Singularity, an alternative containerisation tool, does not assume that you have a system where you are the root user. This has led to increased popularity in HPC contexts due to increased access rights flexibility.

After installing Singularity, VEP may be used with Singularity based on the VEP Docker image from DockerHub:

```
singularity pull --name vep.sif docker://ensemblorg/ensembl-vep
```

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

```
mkdir $HOME/vep_data
singularity exec vep.sif vep --dir $HOME/vep_data --help
```

The Ensembl VEP API, plugins and their dependencies (e.g. Perl APIs, Bio::DB::HTS, htslib, ...) are already installed in the image.

Cache and FASTA files installation

You can run the INSTALL.pl script to install the Cache data and FASTA files. For example, to set up the cache and corresponding FASTA for human GRCh38 in your local folder **`$HOME/vep_data`**:

```
singularity exec vep.sif INSTALL.pl -c $HOME/vep_data -a cf -s homo_sapiens -y GRCh38
```

The installer downloads data to the specified directory (e.g., **`$HOME/vep_data`**). When running VEP via Singularity, point to this directory using **`--dir`**:

```
singularity exec vep.sif vep --dir $HOME/vep_data -i examples/homo_sapiens_GRCh38.vcf --cache
```

Running VEP with data from local folder

Here is an example on running VEP with data from folder **`$HOME/vep_data`** in the local machine (provided that the cache has been downloaded to that folder):

```
singularity exec vep.sif \
  vep --dir $HOME/vep_data \
      --cache --offline --format vcf --vcf --force_overwrite \
      --input_file input/my_input.vcf \
      --output_file output/my_output.vcf \
      --custom file=custom/my_extra_data.bed,short_name=BED_DATA,format=bed,type=exact,coords=1 \
      --plugin NMD
```

Update from a previous version

1. Update your docker container

```
singularity pull --name vep.sif docker://ensemblorg/ensembl-vep
```

2. Update your cache

```
# Install the new cache through the VEP INSTALL.pl script (see "Cache installation" section
above)
singularity exec vep.sif INSTALL.pl -c $HOME/vep_data -a c

# Or install the cache manually
cd $HOME/vep_data
curl -O https://ftp.ensembl.org/pub/release-
114/variation/vep/homo_sapiens_vep_114_GRCh38.tar.gz
tar xzf homo_sapiens_vep_114_GRCh38.tar.gz
```

Nextflow

We offer a Nextflow VEP pipeline that aims to run VEP using simple parallelisation. The pipeline is deployable on an individual Linux machine or on computing clusters running LSF, SLURM or other workload managers.

The process can be summarised briefly by the following steps:

- Splitting the input data into multiple files using a given number of bins
- Running VEP on the split files in parallel
- Merging VEP outputs into a single file

To run the pipeline in a system with Nexflow installed, you will need to prepare a **vep.ini** config file. Here are some examples commands to run the Nextflow VEP pipeline:

```
# Run Nextflow VEP using local VEP installation
# NB: Nextflow automatically downloads the GitHub repository
nextflow run Ensembl/ensembl-vep -r main \
  --input input.vcf \
  --vep_config vep.ini

# Run latest VEP version using Docker
nextflow run Ensembl/ensembl-vep -r main \
  -profile docker \
  --input input.vcf \
  --vep_config vep.ini

# Run VEP 114.0 using Docker
nextflow run Ensembl/ensembl-vep -r main \
  -profile docker \
  --input input.vcf \
  --vep_config vep.ini \
  --vep_version 114.0

# Run VEP 114.0 using SLURM and Singularity
nextflow run Ensembl/ensembl-vep -r main \
  -profile slurm,singularity \
  --input input.vcf \
```

```
  --vep_config vep.ini \
  --vep_version 114.0
```

For a full list of supported profiles, as well as more instructions on setting up and running the pipeline, please refer to the Nextflow VEP instructions.

Variant Effect Predictor — Data formats

Input

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

VEP can use different input formats:

Format	Variant example	Structural variant example
Default VEP input	`1 881907 881906 -/C +`	`1 160283 471362 DUP +`
VCF	`1 65568 . A C . . .`	`1 7936271 . N N[12:58877476[ . . SVTYPE=BND`
HGVS identifiers	`ENST00000618231.3:c.9G>C`	✗ Not supported
Variant identifiers	`rs699`	`nsv1000164`
Genomic SPDI notation	`NC_000016.10:68684738:G:A`	✗ Not supported
REST-style regions	`14:19584687-19584687:-1/T`	`21:25587759-25587769/DEL`

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 (or structural variant type)
5. **strand** - defined as + (forward) or - (reverse). The strand will only be used for VEP to know which alleles to use.
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   +
2   946507    946507    G/C   +
5   140532    140532    T/C   +
8   150029    150029    A/T   +    var2
12  1017956   1017956   T/A   +
14  19584687  19584687  C/T   -
19  66520     66520     G/A   +    var1
```

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/- -
```

Structural variants are also supported by indicating a structural variant type instead of the allele:

```
1    20000     30000     CN4   +  cnv4
1    160283    471362    DUP   +  dup
1    1385015   1387562   DEL   +  del1
```

```
12   1017956   1017956   INV  +  inv
21   25587759  25587769  CN0  +  del2
```

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:

```
#CHROM  POS        ID      REF  ALT            QUAL  FILTER  INFO  FORMAT
1       65568      .       A    C              .     .       .     .
1       230710048  rs699   A    G              .     .       .     .
2       265023     .       C    T              .     .       .     .
3       319780     .       GA   G              .     .       .     .
20      3          .       C    CAAG,CAAGAAG   .     PASS    .     .
21      43762120   rs1300  T    A,C,G          .     .       .     .
```

Structural variants are also supported depending on structural variant type.

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.

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:

```
ENST00000618231.3:c.9G>C
ENST00000471631.1:c.28_33delTCGCGG
ENST00000285667.3:c.1047_1048insC
5:g.140532G>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_005239.6:c.190G>A
NM_001025204.2: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
```

Ambiguous gene-based descriptions

It is possible to use ambiguous descriptions listing only gene symbol or UniProt accession and protein change (e.g. PHF21B:p.Tyr124Cys, P01019:p.Ala268Val), as seen in the literature, though this is not recommended as it can produce multiple different variants at genomic level. The transcripts for a gene are considered in the following order:

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

and the first compatible transcript is used to map variants to the genome for annotation.

Variant identifiers

These should be dbSNP rsIDs (such as `rs699`), or any synonym for a variant present in the Ensembl Variation database. Structural variant identifiers (like `nsv1000164` and `esv1850194`) are also supported.

See here for a list of identifier sources in Ensembl.

Examples:

```
rs1156485833
rs1258750482
rs867704559
esv1815690
nsv1000164
```

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). In SPDI notation, the position refers to the base before the variant, not the base of the variant itsef.

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, 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/-
```

Structural variants are also supported by indicating a structural variant type in the place of the `[allele]`:

```
# structural variant: deletion
21:25587759-25587769/DEL

# structural variant: inversion
21:25587759-25587769/INV
```

Structural variant types

VEP can also call consequences on structural variants using the following input formats:

- Default VEP input
- REST-style regions
- Variant identifiers
- VCF

To recognise a variant as a structural variant, the allele string (or `SVTYPE` in the INFO column of the VCF format) must be set to one of the currently supported values:

- **INS** - insertion
  - **INS:ME** - insertion of mobile element
  - **INS:ME:ALU** - insertion of ALU element
  - **INS:ME:HERV** - insertion of HERV element
  - **INS:ME:LINE1** - insertion of LINE1 element
  - **INS:ME:SVA** - insertion of SVA element

- **DEL** - deletion
  - **DEL:ME** - deletion of mobile element
  - **DEL:ME:ALU** - deletion of ALU element
  - **DEL:ME:HERV** - deletion of HERV element
  - **DEL:ME:LINE1** - deletion of LINE1 element
  - **DEL:ME:SVA** - deletion of SVA element
- **DUP** - duplication
  - **DUP:TANDEM** - tandem duplication
  - **TDUP** - tandem duplication
- **INV** - inversion
- **CNV** - copy number variation
  - The copy number value can be specified like so:
    - `CN0`
    - `CN=4`
    - `CN3,CN4,CN6`
    - `CN=0,CN=2,CN=4`
  - **CNV:TR** - tandem repeats
    - Requires **`INFO`** fields describing the tandem repeat, such as **`RUS`** and **`RN`** – check VCF 4.4 specification, section 5.7
    - Currently, the **`CIRUC`** and **`CIRB`** **`INFO`** fields are ignored when calculating alternative alleles in tandem repeats
- **BND** - chromosome breakpoints
  - Breakpoint variants are composed by one or more breakends
  - In VCF, breakend replacements are inserted into the **`ALT`** column and need to meet the HTS specifications, such as `TG[12:58877476[`
  - Single breakends can be specified in **`ALT`**, such as `T.` and `.G`
  - Multiple, comma-separated alternative breakends can be specified in **`ALT`**, such as `A[22:22893780[,A[X:10932343[`

More information on how VEP processes structural variants can be found here.

Examples of structural variants encoded in VCF format

```
#CHROM  POS       ID   REF  ALT             QUAL  FILTER  INFO
1       160283    dup  .    <DUP>           .     .       SVTYPE=DUP;END=471362
1       1385015   del  .    <DEL>           .     .       SVTYPE=DEL;END=1387562
1       7936271   bnd  N    N[12:58877476[  .     .       SVTYPE=BND
```

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

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** - identifier of any existing variants. Switch on with --check_existing
14. **Extra** - this column contains extra information as key=value pairs separated by ";", see below.

Other output fields:

- **REF_ALLELE** - the reference allele (after minimisation)
- **UPLOADED_ALLELE** - the uploaded allele string (before minimisation)
- **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. Value must be greater than 0.
- **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. *Note: DISTANCE of 0 is possible for insertions happening just before or after a transcript because variant coordinates are considered to be the flanking bases where insertion happens.*
- **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
- **gnomADe_AF** - Frequency of existing variant in gnomAD exomes combined population
- **gnomADe_AFR_AF** - Frequency of existing variant in gnomAD exomes African/American population
- **gnomADe_AMR_AF** - Frequency of existing variant in gnomAD exomes American population
- **gnomADe_ASJ_AF** - Frequency of existing variant in gnomAD exomes Ashkenazi Jewish population
- **gnomADe_EAS_AF** - Frequency of existing variant in gnomAD exomes East Asian population
- **gnomADe_FIN_AF** - Frequency of existing variant in gnomAD exomes Finnish population
- **gnomADg_MID_AF** - Frequency of existing variant in gnomAD exomes Mid-eastern population
- **gnomADe_NFE_AF** - Frequency of existing variant in gnomAD exomes Non-Finnish European population
- **gnomADe_REMAINING_AF** - Frequency of existing variant in gnomAD exomes combined remaining combined populations
- **gnomADe_SAS_AF** - Frequency of existing variant in gnomAD exomes South Asian population
- **gnomADg_AF** - Frequency of existing variant in gnomAD genomes combined population
- **gnomADg_AFR_AF** - Frequency of existing variant in gnomAD genomes African/American population
- **gnomADg_AMI_AF** - Frequency of existing variant in gnomAD genomes Amish population
- **gnomADg_AMR_AF** - Frequency of existing variant in gnomAD genomes American population
- **gnomADg_ASJ_AF** - Frequency of existing variant in gnomAD genomes Ashkenazi Jewish population
- **gnomADg_EAS_AF** - Frequency of existing variant in gnomAD genomes East Asian population
- **gnomADg_FIN_AF** - Frequency of existing variant in gnomAD genomes Finnish population
- **gnomADg_MID_AF** - Frequency of existing variant in gnomAD genomes Mid-eastern population
- **gnomADg_NFE_AF** - Frequency of existing variant in gnomAD genomes Non-Finnish European population
- **gnomADg_REMAINING_AF** - Frequency of existing variant in gnomAD genomes combined remaining combined populations
- **gnomADg_SAS_AF** - Frequency of existing variant in gnomAD genomes 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
- **GENCODE_PRIMARY** - Reports if transcript belongs to GENCODE primary subset
- **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  -  SIFT=deleterious(0);PolyPhen=unknown(0)
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              -    -    -    -    -         -  HGVSc=ENST00000529937.1:c.136-346G>A
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 v114.0
## Output produced at 2017-03-21 14:51:27
## Connected to homo_sapiens_core_114_38 on ensembldb.ensembl.org
## Using cache in /homes/user/.vep/homo_sapiens/114_GRCh38
## Using API version 114, DB version 114
## 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_position  CDS_position  Protein_position  Amino_acids
Codons   Existing_variation  IMPACT    DISTANCE  STRAND  FLAGS
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000525319  Transcript
missense_variant                     742           716           239               S/I
aGc/aTc  -                  MODERATE  -         -1      -
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000534381  Transcript
downstream_gene_variant              -             -             -                 -
-        -                  MODIFIER  1674      -1      -
```

```
11_224088_C/A        11:224088  A       ENSG00000142082  ENST00000529055  Transcript
downstream_gene_variant                -                -                -                -
-        -                   MODIFIER  134       -1      -
11_224585_G/A        11:224585  A       ENSG00000142082  ENST00000529937  Transcript
intron_variant,NMD_transcript_variant  -                -                -                -
-        -                   MODIFIER  -         -1      -
```

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_po
sition|CDS_position|Protein_position|Amino_acids|Codons|Existing_variation|DISTANCE|STRAND|FLAGS|S
YMBOL_SOURCE|HGNC_ID
```

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|Feature_type|Feature|BIOTYPE|EXON|INTRON|HGVSc|HGVSp|cDNA_po
sition|CDS_position|Protein_position">
#CHROM  POS       ID          REF  ALT  QUAL  FILTER  INFO
21      26978790  rs75377686  T    C    .     .
CSQ=C|missense_variant|MODERATE|MRPL39|ENSG00000154719|Transcript|ENST00000419219|protein_coding|2
/8||ENST00000419219.1:c.251A>G|ENSP00000404426.1:p.Asn84Ser|260|251|84
```

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. When using a cache file, frequencies for co-located variants are reported by default (see --af_1kg, --af_gnomade).

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


```
{
  "input": "1 1918090 test1 A G . . .",
  "id": "test1",
  "seq_region_name": "1",
  "start": 1918090,
  "end": 1918090,
  "strand": 1,
  "allele_string": "A/G",
  "most_severe_consequence": "missense_variant",
  "colocated_variants": [
    {
      "id": "COSV57068665",
      "seq_region_name": "1",
      "start": 1918090,
      "end": 1918090,
      "strand": 1,
      "allele_string": "COSMIC_MUTATION"
    },
    {
      "id": "rs28640257",
      "seq_region_name": "1",
      "start": 1918090,
      "end": 1918090,
      "strand": 1,
      "allele_string": "A/G/T",
      "minor_allele": "G",
      "minor_allele_freq": 0.352,
      "frequencies": {
        "G": {
          "amr": 0.5072,
          "gnomade_sas": 0.3635,
          "gnomade": 0.481,
          "gnomade_remaining": 0.4536,
          "gnomade_asj": 0.3939,
          "gnomade_nfe": 0.5042,
          "gnomade_afr": 0.0975,
          "afr": 0.053,
          "gnomade_amr": 0.5568,
          "gnomade_fin": 0.4751,
          "sas": 0.3906,
          "gnomade_eas": 0.4516,
          "eur": 0.4901,
          "eas": 0.4623,
          "gnomade_mid: "0.3306"
        }
      }
    }
  ],
  "transcript_consequences": [
    {
      "variant_allele": "G",
      "consequence_terms": [
        "missense_variant"
      ],
      "gene_id": "ENSG00000178821",
      "transcript_id": "ENST00000310991",
      "strand": -1,
      "cdna_start": 436,
      "cdna_end": 436,
      "cds_start": 422,
      "cds_end": 422,
      "protein_start": 141,
```

```
      "protein_end": 141,
      "codons": "aTg/aCg",
      "amino_acids": "M/T",
      "polyphen_prediction": "benign",
      "polyphen_score": 0.001,
      "sift_prediction": "tolerated",
      "sift_score": 0.22,
      "hgvsp": "ENSP00000311122.3:p.Met141Thr",
      "hgvsc": "ENST00000310991.8:c.422T>C"
      }
  ],
  "regulatory_feature_consequences": [
    {
      "variant_allele": "G",
      "consequence_terms": [
        "regulatory_region_variant"
      ],
      "regulatory_feature_id": "ENSR00000000255"
    }
  ]
}
```


In accordance with JSON conventions, all keys (except alleles) 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
*_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, please open the file in your web browser.

- To prevent VEP writing a stats file, use --no_stats.
- To get a machine-readable text file in place of the HTML file, use --stats_text. You can get both a HTML file and plain text file by using both --stats_text and --stats_html.

- To change the name of the stats file from the default, use --stats_file [file].

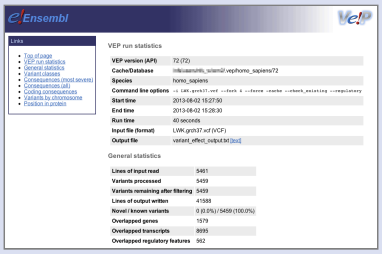
The page contains several sections:

General statistics

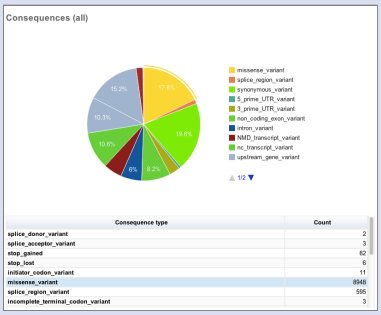
This section contains two tables. The first describes the cache and/or database used, the version of VEP, species, command line parameters, input/output files and run time. The second table contains information about the number of variants, and the number of genes, transcripts and regulatory features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

Charts and tables

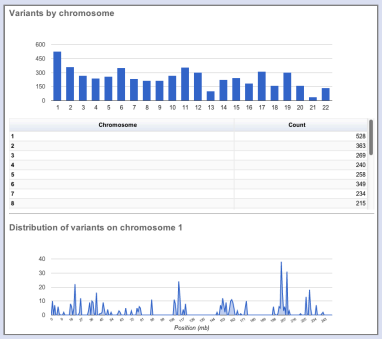
There then follows several charts, most with accompanying tables. Tables and charts are interactive; clicking on a row to highlight it in the table will highlight the relevant segment in the chart, and vice versa.



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
```

VEP can be run in the following modes:

- 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.
- By 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 using your own species and assembly, please use a --fasta file and --gff or --gtf annotation.

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**.

VEP options can also be read from:

- **Configuration files** using --config. Options set in configuration files are overriden if specified on the command line.
- **Environment variables** that start with prefix `VEP_`. For instance, you can set the cache flag with `export VEP_CACHE=1` and the input flag with `export VEP_INPUT="/path/to/input.txt"` before running `./vep`. Options set in environment variables are overriden if specified in configuration files or on the command line.

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, --hgvs, --symbol, --numbers, --domains, --regulatory, --canonical, --protein, --biotype, --af, --af_1kg, --af_esp, --af_gnomade, --af_gnomadg, --max_af, --pubmed, --uniprot, --mane, --tsl, --appris, --variant_class, --gene_phenotype, --mirna
--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*
--no_stats		Don't generate a stats file. Provides marginal gains in run time.
--stats_file [filename]	--sf	Summary stats file name. This file contains a summary of the VEP run. If stats are returned in an HTML file (default), the filename should end in `.html` or `.htm`. *Default = "variant_effect_output.txt_summary.html"*
--stats_html		Generate a HTML stats file (default).
--stats_text		Generate a plain text stats file. Can be combined with `--stats_html` to generate both plain text and HTML stats files.
--warning_file [filename]		File name to write warnings and errors to. *Default = STDERR (standard error)*
--skipped_variants_file [filename]		File name to write skipped variants to. *Default = STDERR (standard error)*
--max_sv_size		Extend the maximum Structural Variant size VEP can process. *Default = 10000000*
--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*

`--safe`		By default, a VEP run is successful even when a plugin reports issues. Use this flag to ensure VEP fails if a plugin raises warnings or generates compilation errors. This is particularly useful to ensure plugins run successfully when using VEP in pipelines. *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 --lrg
`--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 --gencode_primary --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 file=[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 --summary --most_severe --ga4gh_vrs
`--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. For an insertion/deletion, the allele is minimised by default. To access the input allele before minimisation, use --uploaded_allele. *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 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*	NEAREST	
`--distance [bp_distance(,downstream_distance)]`	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	`--minimal` `--individual_zyg`
`--individual_zyg [all|ind list]`	Consider alternate and reference 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. Returns a list of individuals and their zygosity. Only works with VCF files containing individual genotype data; individual IDs are taken from column headers. *Not used by default*	ZYG	`--individual`
`--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 (after minimisation). Mainly useful for the VEP "default" and tab-delimited output formats. *Not used by default*	REF_ALLELE	

`--uploaded_allele`		Adds the uploaded allele string in the output (before minimisation).	UPLOADED_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	--most_severe --summary
`--mirna`		Reports where the variant lies in the miRNA secondary structure (only for Ensembl/GENCODE transcripts). *Not used by default*		
`--no_escape`		Don't URI escape HGVS strings. *Default = escape*		
`--keep_csq`		Don't overwrite existing CSQ entry in VCF INFO field. *Overwrites by default*		
`--vcf_info_field [CSQ|ANN|(other)]`		Change the name of the INFO key that VEP write the consequences to in its VCF output. Use "ANN" for compatibility with other tools such as snpEff. *Default: CSQ*		
`--terms [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_hgvs
`--shift_genomic [0|1]`		Right aligns all variants, including intergenic variants, before consequence calculation and updates the *Location* field. An example using this option can be found here. *Default = 0*		--shift_hgvs
`--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	
`--hgvsg_use_accession`		Force --hgvsg to return RefSeq reference sequence. For example, reports NC_000002.11 for human chromosome 2 (build GRCh38).	HGVSg	
`--hgvsp_use_prediction`		Force --hgvs to return the HGVSp notation in predicted format. For example, ENSP00000233741.4:p.Thr367AsnfsTer13 will be returned as ENSP00000233741.4:p.(Thr367AsnfsTer13).	HGVSp	
`--ambiguous_hgvs [0|1]`		Allow input HGVSp to resolve to all genomic locations. Otherwise, most likely transcript will be selected. *Default: 0 (most likely transcript selected)*		

`--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	
`--ga4gh_vrs`	Add GA4GH Variation Representation Specification (VRS) notation. To generate GA4GH VRS when using --cache or --offline you must use a FASTA file and --fasta. *Not used by default*	GA4GH_VRS	--vcf
`--shift_hgvs [0|1]`	Enable or disable 3' shifting of HGVS notations. HGVS nomenclature requires an ambiguous sequence change to be described at the most 3' possible location. When enabled, this causes "shifting" to the 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. If HGVS_OFFSET is equals to 0, no value will be added to HGVS_OFFSET column. To disable the changing of location at transcript level set --shift_hgvs to 0. *Default: 1 (shift)*		--shift_3prime --shift_genomic
`--transcript_version`	Add version numbers to Ensembl transcript identifiers		
`--gene_version`	Add version numbers to Ensembl gene 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. Some gene symbol, e.g. HGNC, are only available in merged and Ensembl caches and therefore should **not** be used with the --refseq cache option. *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, UNIPROT_ISOFORM	--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 or MANE Plus Clinical transcript for the gene. If --cache or --database annotation source is used, the alternative transcript stable ID is also added. *Not used by default*	MANE, MANE_SELECT, MANE_PLUS_CLINICAL	--most_severe --summary
`--mane_select`	Adds a flag indicating if the transcript is the MANE Select transcript for the gene. If --cache or --database annotation source is used, the alternative transcript stable ID is also added. *Not used by default*	MANE, MANE_SELECT	--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 significance values at the given locus. *Default: 1 (Provide allele-specific annotations)*	CLIN_SIG	
`--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_PO	--database

			PS	
--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. *Deprecated.*	AA_AF, EA_AF	--database
--af_gnomade	--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*	gnomADe_AF, gnomADe_AFR_AF, gnomADe_AMR_AF, gnomADe_ASJ_AF, gnomADe_EAS_AF, gnomADe_FIN_AF, gnomADe_NFE_AF, gnomADe_OTH_AF, gnomADe_SAS_AF	--database --af_gnomad
--af_gnomadg		Include allele frequency from Genome Aggregation Database (gnomAD) genome populations. Note only data from the gnomAD genomes are included; to retrieve data from the additional genomes data set, see this guide. Must be used with --cache *Not used by default*	gnomADg_AF, gnomADg_AFR_AF, gnomADg_AMI_AF, gnomADg_AMR_AF, gnomADg_ASJ_AF, gnomADg_EAS_AF, gnomADg_FIN_AF, gnomADg_MID_AF, gnomADg_NFE_AF, gnomADg_OTH_AF, gnomADg_SAS_AF	--database
--af_exac		Include allele frequency from ExAC project populations. Must be used with --cache. *Deprecated.*	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 co-located 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*		--gencode_primary --refseq
--gencode_primary		Limit your analysis to transcripts belonging to the GENCODE primary set. This set covers all human exons in a minimal set of transcripts. *Not used by default*		--gencode_basic --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]_"` or, a list of stable ids in file acting as a allowlist or a blocklist: `--transcript_filter "not stable_id in blocklist.txt"`		
--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. Checking is done on the minimised sequence. Example chr13 32900399 . AGT A . the As are removed and the reference sequence is checked from 32900400 to see if it matches GT*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: *mane_select, mane_plus_clinical, canonical, appris, tsl, biotype, ccds, rank, length, ensembl, refseq*. 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_all

			ele -- no_intergenic --numbers --pick --pick_allele --polyphen --protein --sift --summary --symbol --tsl --uniprot --xref_refseq --mane -- mane_select --vcf
`--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_all ele -- most_severe -- no_intergenic --numbers --pick --pick_allele --polyphen --protein --sift --symbol --tsl --uniprot --xref_refseq --mane -- mane_select --vcf
`--` `flag_gencode_primary`	Flags transcripts as GENCODE primary using a boolean value. *Not used by default*	GENCODE_ PRIMARY	
`--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 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*	FREQS
`--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
gnomADe	gnomAD exomes combined population
gnomADe_AFR	gnomAD exomes African/African American population
gnomADe_AMR	gnomAD exomes Latino population
gnomADe_ASJ	gnomAD exomes Ashkenazi Jewish population
gnomADe_EAS	gnomAD exomes East Asian population
gnomADe_FIN	gnomAD exomes Finnish population
gnomADe_NFE	gnomAD exomes non-Finnish European population
gnomADe_OTH	gnomAD exomes other population
gnomADe_SAS	gnomAD exomes South Asian population
gnomADg	gnomAD genomes combined population
gnomADg_AFR	gnomAD genomes African/African American population
gnomADg_AMR	gnomAD genomes Latino population
gnomADg_AMI	gnomAD genomes Amish population
gnomADg_ASJ	gnomAD genomes Ashkenazi Jewish population
gnomADg_EAS	gnomAD genomes East Asian population
gnomADg_FIN	gnomAD genomes Finnish population
gnomADg_MID	gnomAD genomes Mid-eastern population
gnomADg_NFE	gnomAD genomes non-Finnish European population
gnomADg_OTH	gnomAD genomes other population
gnomADg_SAS	gnomAD genomes 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 --af_gnomade --af_gnomadg --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
  - Requires a FASTA file in --offline mode or if the desired species or assembly is not part of the Ensembl species list.
- 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 (https://ftp.ensembl.org/pub/release-114/variation/indexed_vep_cache/)

Essential for human and other species with large sets of variant data - requires Bio::DB::HTS (setup by INSTALL.pl) or tabix, e.g.:

```
cd $HOME/.vep
curl -O https://ftp.ensembl.org/pub/release-
114/variation/indexed_vep_cache/homo_sapiens_vep_114_GRCh38.tar.gz
tar xzf homo_sapiens_vep_114_GRCh38.tar.gz
```

FTP directories with indexed VEP cache data:

Ensembl:	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.

Pangenome and alternative assemblies

VEP caches are also available for Human Pangenome Reference Consortium (HPRC) data at the Ensembl HPRC data page. Click here for more information on how to use VEP with HPRC data.

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 114.

Source	Version (GRCh38)	Version (GRCh37)
Ensembl database version	114	114
Genome assembly	GRCh38.p14	GRCh37.p13
MANE Version	v1.4	n/a
GENCODE	48	19
RefSeq	GCF_000001405.40-RS_2023_10 (GCF_000001405.40_GRCh38.p14_genomic.gff)	105.20220307 (GCF_000001405.25_GRCh37.p13_genomic.gff)
Regulatory build	1.0	1.0
PolyPhen	2.2.3	2.2.2
SIFT	6.2.1	5.2.2
dbSNP	156	156
COSMIC	100	98
HGMD-PUBLIC	2020.4	2020.4
ClinVar	2024-09	2023-06
1000 Genomes	Phase 3 (remapped)	Phase 3
gnomAD exomes	v4.1	v4.1
gnomAD genomes	v4.1	v4.1

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. This allows you to run VEP on data from any species and assembly.

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 flag --gff :

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

  NOTE: If you wish to customise the name of the GFF as it appears in the SOURCE field and VEP output header, use the longer --custom annotation form:

  ```
  --custom file=data.gff.gz,short_name=frequency,format=gff
  ```

- **GTF file**

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

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

  NOTE: If you wish to customise the name of the GFF as it appears in the SOURCE field and VEP output header, use the longer --custom annotation form:

  ```
  --custom file=data.gtf.gz,short_name=frequency,format=gtf
  ```

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.

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

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

Lines of other types will be ignored; if this leads to an incomplete transcript model, the whole transcript model may be discarded. If unsupported types are used you will see a warning like the following -

```
WARNING: Ignoring 'five_prime_utr' feature_type from Homo_sapiens.GRCh38.111.gtf.gz GFF/GTF file.
This feature_type is not supported in VEP.
```

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"; transcript_biotype "protein_coding";
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/114_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)
- Construct transcript models from a GFF or GTF file without accessing a database (specially useful for performance reasons or if using data from species/assembly not part of Ensembl species list)

FASTA files from Ensembl 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.

The following plugins do require the fasta file to be explicitly passed as a command line argument (i.e. `--fasta /VEP_DIR/your_downloaded.fasta`)

- CSN
- GeneSplicer
- MaxEntScan

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 https://ftp.ensembl.org/pub/release-
114/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_114_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_114_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_114_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_114_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_cache}->{peptide}))
```

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 **that must be separated by whitespace**:

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:

```
# filter for MANE transcripts
--filter "MANE"

# 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 annotation formats:

Format	Type	Description	Notes
GFF GTF	Gene/transcript annotations	Formats to describe genes and other genomic features — format specifications: GFF3 and GTF	Requires a **FASTA** file in offline mode or if the desired species or assembly is not part of the Ensembl species list.
VCF	Variant data	A format used to describe genomic variants	VEP uses the 3rd column as the identifier. INFO and FILTER fields from records may be added to the VEP output.
BED	Basic/uninterpreted data	A simple tab-delimited format containing 3-12 columns of data. The first 3 columns contain the coordinates of the feature.	VEP uses the 4th column (if available) as the feature identifier.
bigWig	Basic/uninterpreted data	A format for storage of dense continuous data.	VEP uses the value for the given position as the identifier. 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

Using positional options in `--custom` with VEP 109 and earlier (compatible with VEP 114)

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 or specify "FILTER" for the FILTER field, 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. Similarly FILTER field will appear as "test_FILTER".
  - 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
https://ftp.ensembl.org/pub/data_files/homo_sapiens/GRCh37/variation_genotype/TOPMED_GRCh37.vcf.gz
,,vcf,exact,0,TOPMED

    # For multiple custom files, use:
    ./vep [...] --custom clinvar.vcf.gz,ClinVar,vcf,exact,0,CLNSIG,CLNREVSTAT,CLNDN \
                --custom TOPMED_GRCh38_20180418.vcf.gz,topmed_20180418,vcf,exact,0,TOPMED \
                --custom UK10K_COHORT.20160215.sites.GRCh38.vcf.gz,uk10k,vcf,exact,0,AF_ALSPAC
```

Using key-value pairs in `--custom` with VEP 114

Since VEP 110, you can configure each custom file using a comma-separated list of key-value pairs:

```
./vep [...] --custom
file=Filename,short_name=Short_name,format=File_type,type=Annotation_type,fields=VCF_fields
```

The order of the options is irrelevant and most options have sensible defaults as described below:

Option	Accepted values	Description
`file`	String with valid path to file	**(Required) Filename:** The path to the file. For Tabix indexed files, VEP will check if both the file and the corresponding index (`.tbi`) exist. For remote files, VEP will check that the tabix index is accessible on startup.
`format`	`bed`, `gff`, `gtf`, `vcf` or `bigwig`	**(Required) File format** of `file`.
`short_name`	Annotation filename (default) or any string without commas	**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.
`fields`		**VCF fields:** Percentage (%) separated list of INFO fields to print (such as `AC`) present in the custom input VCF or specify `FILTER` for the FILTER field, 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. Similarly FILTER field will appear as `test_FILTER`. • 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.
`type`	`overlap` (default), `within`, `surrounding` or `exact`	**Annotation type:** • `overlap`: reports any annotation that overlaps the variant by even 1 base pair. • `within` (*): only reports annotations within the variant. • `surrounding` (*): only reports annotations that completely surround the variant. • `exact`: only reports annotations whose coordinates match exactly those of the variant. This is suitable for position-specific information such as conservation scores, allele frequencies or phenotype information.
`overlap_cutoff`	From `0` (default) to `100`	**Minimum percentage overlap (*)** between annotation and variant. See also `reciprocal`.
`reciprocal`	`0` (default) or `1`	**Mode of calculating the overlap percentage (*):** • `0`: percentage of annotation covered by variant • `1`: percentage of variant covered by annotation
`distance`	`0` or a positive integer (disabled by default)	Distance (in base pairs) to the ends of the overlapping feature (*).
`coords`	`0` (default) or `1`	**Force report coordinates:** • `0`: outputs the identifier field (or value in the case of `bigWig`) if available; otherwise, outputs coordinates instead. • `1`: always outputs the coordinates of an overlapping custom feature.
`same_type`	`0` (default) or `1`	Only match identical variant classes (*). For instance, only match deletions with deletions. This is only available for VCF annotations.

`num_records`	`50` (default), `all`, `0` or any positive integer	**Number of matching records to display.** Any remaining records are represented with ellipsis (`...`). Use `num_records = all` to display all matching records and `num_records = 0` to only display `...` if there are matching records.
`summary_stats`	`none` (default), `min`, `mean`, `max`, `count` or `sum`	**Summary statistics to display**. A percentage-separated list may be used to calculate multiple summary statistics, such as `min%mean%max%count%sum`.

When `format = vcf`, the features marked with (*) only work on structural variants.

Examples:

```
# BigWig file
./vep [...] --custom file=frequencies.bw,short_name=Frequency,format=bigwig,type=exact,coords=0
# BED file
./vep [...] --custom
file=http://www.myserver.com/data/myPhenotypes.bed.gz,short_name=Phenotype,format=bed,type=exact,c
oords=1
# VCF file
./vep [...] --custom
file=https://ftp.ensembl.org/pub/data_files/homo_sapiens/GRCh37/variation_genotype/TOPMED_GRCh37.v
cf.gz,format=vcf,type=exact,coords=0,fields=TOPMED
./vep [...] --custom
file=gnomad_v2.1_sv.sites.vcf.gz,short_name=gnomad,fields=PC%EVIDENCE%SVTYPE,format=vcf,type=withi
n,reciprocal=1,overlap_cutoff=80

# For multiple custom files, use:
./vep [...] --custom
file=clinvar.vcf.gz,short_name=ClinVar,format=vcf,type=exact,coords=0,fields=CLNSIG%CLNREVSTAT%CLN
DN \
            --custom
file=TOPMED_GRCh38_20180418.vcf.gz,short_name=topmed_20180418,format=vcf,type=exact,coords=0,field
s=TOPMED \
            --custom
file=UK10K_COHORT.20160215.sites.GRCh38.vcf.gz,short_name=uk10k,format=vcf,type=exact,coords=0,fie
lds=AF_ALSPAC
```

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

2. Example of command you can use:

```
./vep [...] --custom
file=clinvar.vcf.gz,short_name=ClinVar,format=vcf,type=exact,coords=0,fields=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 --custom
file=/path/to/custom_files/clinvar.vcf.gz,short_name=ClinVar,format=vcf,type=exact,coords=0,fie
lds=CLNSIG%CLNREVSTAT%CLNDN
```

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;STRAND=-1;ClinVar=18068;ClinVar_CLNDN=Hypertension,_essential,_susceptibility_t
o|Preeclampsia,_susceptibility_to|Renal_dysplasia|Susceptibility_to_progression_to_renal_failur
e_in_IgA_nephropathy|not_specified;ClinVar_CLNREVSTAT=criteria_provided,_multiple_submitters,_n
o_conflicts;ClinVar_CLNSIG=Benign;ClinVar_FILTER=.
1_230710048_A/G      1:230710048  G       ENSG00000244137  ENST00000412344  Transcript
downstream_gene_variant  ...
IMPACT=MODIFIER;DISTANCE=650;STRAND=-1;ClinVar=18068;ClinVar_CLNDN=Hypertension,_essential,_sus
ceptibility_to|Preeclampsia,_susceptibility_to|Renal_dysplasia|Susceptibility_to_progression_to
_renal_failure_in_IgA_nephropathy|not_specified;ClinVar_CLNREVSTAT=criteria_provided,_multiple_
submitters,_no_conflicts;ClinVar_CLNSIG=Benign;ClinVar_FILTER=.
```

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|Feature_type|Feature|BIOTYPE|EXON|INTRON|HGVSc|HGVSp|cDNA
_position|CDS_position|Protein_position|Amino_acids|Codons|Existing_variation|DISTANCE|STRAND|F
LAGS|SYMBOL_SOURCE|HGNC_ID|SOURCE|ClinVar|ClinVar_CLNSIG|ClinVar_CLNREVSTAT|ClinVar_CLNDN">
##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|ENST00000366667|protein_coding|2
```

```
/5||||1018|803|268|M/T|aTg/aCg|||-1||HGNC|HGNC:333||18068|Benign|criteria_provided&_multiple_su
bmitters&_no_conflicts|Hypertension&_essential&_susceptibility_to&Preeclampsia&_susceptibility_
to&Renal_dysplasia&Susceptibility_to_progression_to_renal_failure_in_IgA_nephropathy¬_specified
,G|downstream_gene_variant|MODIFIER|AL512328.1|ENSG00000244137|Transcript|ENST00000412344|antis
ense||||||||||650|-1||Clone_based_ensembl_gene|||18068|Benign|criteria_provided&_multiple_subm
itters&_no_conflicts|Hypertension&_essential&_susceptibility_to&Preeclampsia&_susceptibility_to
&Renal_dysplasia&Susceptibility_to_progression_to_renal_failure_in_IgA_nephropathy&not_specifie
d
```

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.

Example - phyloP and phastCons conservation scores

The UCSC Genome Browser provides multiple alignment files with phyloP and phastCons conservation scores for different genomes in the BigWig (`.bw`) format.

These files can be remotely used as VEP custom annotations by simply pointing to their URL. For instance, to include phyloP or phastCons 100 way conservation scores found in the Downloads section of the UCSC Genome Browser, you can use commands such as:

```
# Human GRCh38/hg38 phyloP100way scores
./vep [...] --custom
file=http://hgdownload.soe.ucsc.edu/goldenPath/hg38/phyloP100way/hg38.phyloP100way.bw,short_name=p
hyloP100way,format=bigwig

# Human GRCh38/hg38 phastCons100way scores
./vep [...] --custom
file=http://hgdownload.soe.ucsc.edu/goldenPath/hg38/phastCons100way/hg38.phastCons100way.bw,short_
name=phastCons100way,format=bigwig
```

Variant Effect Predictor Plugins

VEP can use plugin modules written in Perl to **extend, filter and manipulate** the VEP output.

To use plugins with VEP, you can:

- Install them using VEP's installer script. You can quickly check installed plugins by running:
  - ```
    perl INSTALL.pl -a p -g list
    ```
- Use Ensembl VEP in Docker and Singularity. VEP plugins and their dependencies are available in the Docker image.
- Use the VEP web and REST interfaces. Not all plugins are available therein and they may have limited options.

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	Category	External libraries	Developer
AlphaMissense	This plugin for the Ensembl Variant Effect Predictor (VEP) annotates missense variants with the pre-computed AlphaMissense pathogenicity scores. AlphaMissense is a deep learning model developed by Google DeepMind that predicts the pathogenicity of single nucleotide missense variants. This plugin will add two annotations per missense variant: • `am_pathogenicity`, a continuous score between 0 and 1 which can be interpreted as the predicted probability of the variant being pathogenic. • `am_class` is the classification of the variant into one of three discrete categories: `likely_pathogenic`, `likely_benign`, or `ambiguous`. These are derived using the following thresholds of am_pathogenicity: `likely_benign` if `am_pathogenicity` < 0.34; `likely_pathogenic` if `am_pathogenicity` > 0.564; `ambiguous` otherwise. These thresholds were chosen to achieve 90% precision for both pathogenic and benign ClinVar variants. Note that AlphaMissense was not trained on ClinVar variants. Variants labeled as `ambiguous` should be treated as `unknown` or `uncertain` according to AlphaMissense. This plugin is available for both GRCh37 (hg19) and GRCh38 (hg38) genome builds. The prediction scores of AlphaMissense can be downloaded from https://console.cloud.google.com/storage/browser/dm_alphamissense (AlphaMissense Database Copyright (2023) DeepMind Technologies Limited). Data contained within the AlphaMissense Database is licensed under the Creative Commons Attribution 4.0 International License (CC-BY) (the “License”). You may obtain a copy of the License at: https://creativecommons.org/licenses/by/4.0/legalcode. Use of the AlphaMissense Database is subject to Google Cloud Platform Terms of Service Please cite the AlphaMissense publication alongside the VEP if you use this resource: https://doi.org/10.1126/science.adg7492	Pathogenicity predictions	-	Ensembl
    ```

Plugin	Description	Category	External libraries	Developer
	Disclaimer: The AlphaMissense Database and other information provided on or linked to this site is for theoretical modelling only, caution should be exercised in use. It is provided "as-is" without any warranty of any kind, whether express or implied. For clarity, no warranty is given that use of the information shall not infringe the rights of any third party (and this disclaimer takes precedence over any contrary provisions in the Google Cloud Platform Terms of Service). The information provided is not intended to be a substitute for professional medical advice, diagnosis, or treatment, and does not constitute medical or other professional advice. Before running the plugin for the first time, you need to create a tabix index (requires tabix to be installed). `tabix -s 1 -b 2 -e 2 -f -S 1 AlphaMissense_hg38.tsv.gz` `tabix -s 1 -b 2 -e 2 -f -S 1 AlphaMissense_hg19.tsv.gz` Options are passed to the plugin as key=value pairs: **Argument** — **Description** `file` — (mandatory) Tabix-indexed AlphaMissense data `cols` — (optional) Colon-separated columns to print from AlphaMissense data; if set to `all`, all columns are printed (default: `Missense_pathogenicity:Missense_class`) `transcript_match` — Only print data if transcript identifiers match those from AlphaMissense data (default: 0) AlphaMissense predictions are matched to input data by genomic location and protein change by default. **Usage examples:** `mv AlphaMissense.pm ~/.vep/Plugins` `# print AlphaMissense scores and predictions (default)` `./vep -i variations.vcf --plugin AlphaMissense,file=/full/path/to/file.tsv.gz` `# print all AlphaMissense information` `./vep -i variations.vcf --plugin AlphaMissense,file=/full/path/to/file.tsv.gz,cols=all` `# only report results for the transcripts in the AlphaMissense prediction` `./vep -i variations.vcf --plugin AlphaMissense,file=/full/path/to/file.tsv.gz,transcript_match=1`			
AncestralAllele	A VEP plugin that retrieves ancestral allele sequences from a FASTA file.	Conservation	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	Ensembl produces FASTA file dumps of the ancestral sequences of key species. • Data files for GRCh37: https://ftp.ensembl.org/pub/release-75/fasta/ancestral_alleles/ • Data files for GRCh38: https://ftp.ensembl.org/pub/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 INSTALL.pl): `wget https://ftp.ensembl.org/pub/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, particularly if you do not have `Bio::DB::HTS` installed. Special cases: • - represents an insertion • ? indicates the chromosome could not be looked up in the FASTA **Usage examples:** `mv AncestralAllele.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin AncestralAllele,homo_sapiens_ancestor_GRCh38.fa.gz`			
AVADA	Automatic VAriant evidence DAtabase is a novel machine learning tool that uses natural language processing to automatically identify pathogenic genetic variant evidence in full-text primary literature about monogenic disease and convert it to genomic coordinates. Please cite the AVADA publication alongside the VEP if you use this resource: https://pubmed.ncbi.nlm.nih.gov/31467448/ NB: The plugin currently does not annotate for downstream_gene_variant and upstream_gene_variant. Pre-requisites 1. AVADA data is available for GRCh37 and can be downloaded from: http://bejerano.stanford.edu/AVADA/avada_v1.00_2016.vcf.gz `wget http://bejerano.stanford.edu/AVADA/avada_v1.00_`	Phenotype data and citations	List::MoreUtils qw(uniq)	Ensembl

Plugin	Description	Category	External libraries	Developer

```
2016.vcf.gz
```

2. The file needs to be tabix indexed. You can do this by following commands:

```
gzip -d avada_v1.00_2016.vcf.gz
bgzip avada_v1.00_2016.vcf
tabix avada_v1.00_2016.vcf.gz
```

3. As you have already noticed, tabix utility must be installed in your path to use this plugin.

The plugin can then be run to retrieve AVADA annotations. By default, the variants are matched with the HGNC gene symbol

```
./vep -i variations.vcf --plugin
AVADA,file=path/to/file
```

The output always includes one of the following columns depending on the option passed:

- `AVADA_PMID`: PubMed ID evidence for the variant as reported by AVADA
- `AVADA_PMID_WITH_VARIANT`: PubMed ID evidence for the variant as reported by AVADA along with the original variant string
- `AVADA_PMID_WITH_FEATURE`: PubMed ID evidence for the variant as reported by AVADA along with feature id
- `AVADA_PMID_WITH_FEATURE_AND_VARIANT`: PubMed ID evidence for the variant as reported by AVADA along with feature id and original variant string

The plugin can optionally be run by specifying the feature to match with.

In order to match by HGNC gene symbol:

```
./vep -i variations.vcf --plugin
AVADA,file=path/to/file,feature_match_by=gene_s
ymbol
```

In order to match by Ensembl gene identifier :

```
./vep -i variations.vcf --plugin
AVADA,file=path/to/file,feature_match_by=ensemb
l_gene_id
```

In order to match by RefSeq identifier :

```
./vep -i variations.vcf --plugin
AVADA,file=path/to/file,feature_match_by=refseq
_id
```

The plugin can also be run to report the original variant string reported in the publication.

```
./vep -i variations.vcf --plugin
AVADA,file=path/to/file,original_variant_string
=1
```

Plugin	Description	Category	External libraries	Developer
	Usage examples: `./vep -i variations.vcf --plugin AVADA,file=path/to/file` `./vep -i variations.vcf --plugin AVADA,file=path/to/file,feature_match_by=<gene_symbol|ensembl_gene_id|refseq_id>`			
BayesDel	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds the BayesDel scores to VEP output. BayesDel is a deleteriousness meta-score combining multiple deleteriousness predictors to create an overall score. It works for coding and non-coding variants, single nucleotide variants and small insertion/deletions. The range of the score is from -1.29334 to 0.75731. The higher the score, the more likely the variant is pathogenic. For more information please visit: https://fenglab.chpc.utah.edu/BayesDel/BayesDel.html Please cite the BayesDel publication alongside the Ensembl VEP if you use this resource: https://onlinelibrary.wiley.com/doi/full/10.1002/humu.23158 BayesDel pre-computed scores can be downloaded from https://drive.google.com/drive/folders/1K4LI6ZSsUGBhHoChUtegC8bgCt7hbQIA Note: These files only contain pre-computed BayesDel scores for missense variants for assembly GRCh37. For GRCh37: `tar zxvf BayesDel_170824_addAF.tgz` `rm *.gz.tbi` `gunzip *.gz` `for f in BayesDel_170824_addAF_chr*; do grep -v "^#" $f >> BayesDel_170824_addAF.txt; done` `cat BayesDel_170824_addAF.txt | sort -k1,1 -k2,2n > BayesDel_170824_addAF_sorted.txt` `grep "^#" BayesDel_170824_addAF_chr1 > BayesDel_170824_addAF_all_scores.txt` `cat BayesDel_170824_addAF_sorted.txt >> BayesDel_170824_addAF_all_scores.txt` `bgzip BayesDel_170824_addAF_all_scores.txt` `tabix -s 1 -b 2 -e 2 BayesDel_170824_addAF_all_scores.txt.gz` For GRCh38: Remap GRCh37 file The tabix utility must be installed in your path to use this plugin. **Usage examples:** `mv BayesDel.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin BayesDel,file=/path/to/BayesDel/BayesDel_170824_addAF_all_scores.txt.gz`	Pathogenicity predictions	-	Ensembl
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. **Usage examples:**	Conservation	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`mv Blosum62.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Blosum62`			
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 SNV and indels data files (and respective Tabix index files) can be downloaded from - http://cadd.gs.washington.edu/download The CADD SV data files (and respective Tabix index files) can be downloaded from - https://kircherlab.bihealth.org/download/CADD-SV/v1.1/ By default the plugin is designed to not annotate SV variant if a SNV and/or indels CADD annotation file is provided. Because it can results in too many lines matched from the annotation files and increase run time exponentially. You can override this behavior by providing force_annotate=1 which will force the plugin to annotate with the expense of increasing runtime. 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. **Usage examples:** `mv CADD.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin CADD,snv=/FULL_PATH_TO_CADD_FILE/whole_genome_SNVs.tsv.gz,indels=/FULL_PATH_TO_CADD_FILE/InDels.tsv.gz` `./vep -i structural_variations.vcf --plugin CADD,sv=/FULL_PATH_TO_CADD_FILE/1000G_phase3_SVs.tsv.gz` `./vep -i structural_variations.vcf --plugin CADD,snv=/FULL_PATH_TO_CADD_FILE/whole_genome_SNVs.tsv.gz,indels=/FULL_PATH_TO_CADD_FILE/InDels.tsv.gz,force_annotate=1`	Pathogenicity predictions	-	Ensembl
CAPICE	A VEP plugin that retrieves CAPICE scores for variants from one or more tabix-indexed CAPICE data files, in order to predict their pathogenicity. Please cite the CAPICE publication alongside the VEP if you use this resource: https://pubmed.ncbi.nlm.nih.gov/32831124/ The tabix utility must be installed in your path to use this plugin. The CAPICE SNVs, InDels and respective index (TBI) files for GRCh37 can be downloaded from https://zenodo.org/record/3928295 To filter results, please use filter_vep with the output file or standard output. Documentation on filter_vep is available at: https://www.ensembl.org/info/docs/tools/vep/script/vep_filter.html For recommendations on threshold selection, please read the CAPICE publication. **Usage examples:**	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`mv CAPICE.pm ~/.vep/Plugins` `# Download CAPICE SNVs, InDels and index (TBI) files to the same path` `# - capice_v1.0_build37_indels.tsv.gz` `# - capice_v1.0_build37_indels.tsv.gz.tbi` `# - capice_v1.0_build37_snvs.tsv.gz` `# - capice_v1.0_build37_snvs.tsv.gz.tbi` `./vep -i variations.vcf --plugin CAPICE,snv=/FULL_PATH_TO_CAPICE_FILE/capice_v1.0_build37_snvs.tsv.gz,indels=/FULL_PATH_TO_CAPICE_FILE/capice_v1.0_build37_indels.tsv.gz` `./filter_vep -i variant_effect_output.txt --filter "CAPICE_SCORE >= 0.02"`			
Carol	A VEP plugin 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: https://sanger.ac.uk/tool/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 73(1):47-51 (2012) doi:10.1159/000334984 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 40(D1):D84-D90 (2011) doi: 10.1093/nar/gkr991 **Usage examples:** `mv Carol.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Carol`	Pathogenicity predictions	Math::CDF qw(pnorm qnorm)	Ensembl
ClinPred	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds pre-calculated scores from ClinPred. ClinPred is a prediction tool to identify disease-relevant nonsynonymous variants. Please cite the ClinPred publication alongside the VEP if you use this resource: https://www.sciencedirect.com/science/article/pii/S0002929718302714 ClinPred scores can be downloaded from https://sites.google.com/site/clinpred/download The following steps are neccessary to tabix the ClinPred.txt.gz file before running the plugin:	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	For GRCh37: `gzip -d ClinPred.txt.gz # to unzip the text file` `cat ClinPred.txt | tr " " "\t" > ClinPred_tabbed.tsv # change to tab-delimited file` `sed -i '1s/.*/#&/' ClinPred_tabbed.tsv # comment the first line` `sed -i 1s/Chr/chr/ ClinPred_tabbed.tsv # convert Chr to chr` `bgzip ClinPred_tabbed.tsv` `tabix -f -s 1 -b 2 -e 2 ClinPred_tabbed.tsv.gz` For GRCh38: `gzip -d ClinPred_hg38.txt.gz # unzip the text file` `awk '($2 == "Start" || $2 ~ /^[0-9]+$/){print $0}' ClinPred_hg38.txt > "ClinPred_hg38_tabbed.tsv" # remove problematic lines` `sed -i '1s/.*/#&/' ClinPred_hg38_tabbed.tsv # comment the first line` `sed -i 1s/Chr/chr/ ClinPred_hg38_tabbed.tsv # convert Chr to chr` { head -n 1 ClinPred_hg38_tabbed.tsv; tail -n +2 ClinPred_hg38_tabbed.tsv | sort -k1,1V -k2,2V; } > ClinPred_hg38_sorted_tabbed.tsv # sort file by chromosome and position `bgzip ClinPred_hg38_sorted_tabbed.tsv` `tabix -f -s 1 -b 2 -e 2 ClinPred_hg38_sorted_tabbed.tsv.gz` The tabix utility must be installed in your path to use this plugin. Check https://github.com/samtools/htslib.git for instructions. **Usage examples:** `mv ClinPred.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin ClinPred,file=ClinPred_tabbed.tsv.gz`			
Condel	A VEP plugin 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 https://bg.upf.edu/condel. 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 by this order: 1. Path to a Condel configuration directory which contains cutoffs and the distribution files, etc.	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	2. Output: output the Condel score (`s`), prediction (`p`) or both (`b`); both (`b`) is the default. 3. Version of Condel to use: either 1 (original version) or 2 (newer version); `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 **Usage examples:** `mv Condel.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Condel,/path/to/config/Condel/config,b`			
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. If the MAX parameter is used, the maximum score of any of the affected bases will be reported instead. 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 https://ftp.ensembl.org/pub/current_compara/conservation_scores/) **Usage examples:** `mv Conservation.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Conservation,mammals` `./vep -i variations.vcf --plugin Conservation,/path/to/bigwigfile.bw` `./vep -i variations.vcf --plugin Conservation,/path/to/bigwigfile.bw,MAX`	Conservation	Net::FTP	Ensembl

Plugin	Description	Category	External libraries	Developer
	``./vep -i variations.vf --plugin Conservation,database,GERP_CONSERVATION_SCORE,mammals``` ```./vep -i variations.vf --plugin Conservation,database,GERP_CONSERVATION_SCORE,mammals,MAX```			
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 • dbNSFP v4 https://www.ncbi.nlm.nih.gov/pubmed/33261662 You must have the `Bio::DB::HTS` module or the tabix utility must be installed in your path to use this plugin. About dbNSFP data files: • Downoad dbNSFP files from https://sites.google.com/site/jpopgen/dbNSFP. • There are two distinct branches of the files provided for academic and commercial usage. Please use the appropriate files for your use case. • The file must be processed depending on dbNSFP release version and assembly (see commands below). We recommend using `-T` option with the sort command to specify a temporary directory with sufficient space. • The resulting file must be indexed with tabix before use by this plugin (see commands below). For release 4.9c: ```version=4.9c wget https://dbnsfp.s3.amazonaws.com/dbNSFP${version}.zip unzip dbNSFP${version}.zip zcat dbNSFP${version}_variant.chr1.gz \| head -n1 > h``` # GRCh38/hg38 data ```zgrep -h -v ^#chr dbNSFP${version}_variant.chr* | sort -k1,1 -k2,2n - | cat h - | bgzip -c > dbNSFP${version}_grch38.gz tabix -s 1 -b 2 -e 2 dbNSFP${version}_grch38.gz``` # GRCh37/hg19 data ```zgrep -h -v ^#chr dbNSFP${version}_variant.chr* | awk '$8 != "." ' \| sort -k8,8 -k9,9n - \| cat h - \| bgzip -c > dbNSFP${version}_grch37.gz tabix -s 8 -b 9 -e 9 dbNSFP${version}_grch37.gz``` When running the plugin you must list at least one column to retrieve from the dbNSFP file, specified as parameters to the plugin,	Pathogenicity predictions	File::Basename qw(basename)	Ensembl

such as:

```
--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 `|` with `&`. However, some data field columns, e.g. `Interpro_domain`, use the replacement characters. We added a file with replacement logic for customising the required replacement of `;` and `|` in dbNSFP data columns. In addition to the default replacements (`;` to `,` and `|` to `&`) users can add 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_repla
cement_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.

The plugin matches rows in the tabix-indexed dbNSFP file on:

- genomic position
- alt allele
- `aaref` - reference amino acid
- `aaalt` - alternative amino acid

To match only on the genomic position and the alt allele use `pep_match=0`:

```
--plugin
dbNSFP,/path/to/dbNSFP.gz,pep_match=0,col1,col2
```

Some fields contain multiple values, one per Ensembl transcript ID. By default all values are returned, separated by `;` in the default VEP output format. To return values only for the matched Ensembl transcript ID use `transcript_match=1`. This behaviour only affects transcript-specific fields; non-transcript-specific fields are unaffected.

```
--plugin
dbNSFP,/path/to/dbNSFP.gz,transcript_match=1,co
```

Plugin	Description	Category	External libraries	Developer
	`l1,col2` NB 1: Using this flag may cause no value to return if the version of the Ensembl transcript set differs between VEP and dbNSFP. NB 2: MutationTaster entries are keyed on a different set of transcript IDs. Using the `transcript_match` flag with any MutationTaster field selected will have no effect i.e. all entries are returned. Information on corresponding transcript(s) for MutationTaster fields can be found using http://www.mutationtaster.org/ChrPos.html **Usage examples:** `mv dbNSFP.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin dbNSFP,/path/to/dbNSFP.gz,col1,col2` `./vep -i variations.vcf --plugin dbNSFP,'consequence=ALL',/path/to/dbNSFP.gz,col1,col2` `./vep -i variations.vcf --plugin dbNSFP,'consequence=3_prime_UTR_variant&intron_variant',/path/to/dbNSFP.gz,col1,col2`			
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.	Splicing predictions	-	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>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:

<pre>--plugin
dbscSNV,http://my.files.com/dbscSNV.txt.gz</pre>
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.

Usage examples:

<pre>mv dbscSNV.pm ~/.vep/Plugins
./vep -i variations.vc f --plugin
dbscSNV,/path/to/dbscSNV1.1_GRCh38.txt.gz</pre></td><td></td><td></td><td></td></tr>
<tr><td>DeNovo</td><td>A VEP plugin that identifies de novo variants in a VCF file. The plugin is not compatible with JSON output format.

Options are passed to the plugin as key=value pairs:

<table><tr><th>Argument</th><th>Description</th></tr><tr><td>ped</td><td>Path to PED file (mandatory) The file is tab or white-space delimited with five mandatory columns:
family ID
individual ID
paternal ID
maternal ID
sex
phenotype (optional)</td></tr><tr><td>report_dir</td><td>Write files in report_dir (optional)</td></tr><tr><td>full_report</td><td>Set to 1 to report all types of variants (optional) By default, the plugin only reports de novo variants.</td></tr></table>
The plugin can then be run:

<pre>./vep -i variations.vc --plugin
DeNovo,ped=samples.ped
./vep -i variations.vc --plugin
DeNovo,ped=samples.ped,report_dir=path/to/dir
./vep -i variations.vc --plugin
DeNovo,ped=samples.ped,report_dir=path/to/dir,f
ull_report=1</pre>
Usage examples:

<pre>mv DeNovo.pm ~/.vep/Plugins
./vep -i variations.vc --plugin
DeNovo,ped=samples.ped
./vep -i variations.vc --plugin
DeNovo,ped=samples.ped,full_report=1</pre></td><td>Variant data</td><td>List::MoreUtils
qw(uniq)
Cwd</td><td>Ensembl</td></tr>
</table>

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td>DosageSensitivity</td><td>A VEP plugin that retrieves haploinsufficiency and triplosensitivity probability scores for affected genes from a dosage sensitivity catalogue published in paper - https://www.sciencedirect.com/science/article/pii/S0092867422007887

Please cite the above publication alongside the VEP if you use this resource.

This plugin returns two scores:

• pHaplo score gives the probability of a gene being haploinsufficient (deletion intolerant)
• pTriplo score gives the probability of a gene being triploinsensitive (duplication intolerant)

Pre-requisites: You need the compressed tsv file containing the dosage sensitivity score. The file Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz can be downloaded from here - https://zenodo.org/record/6347673/files/Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz

Options are passed to the plugin as key=value pairs:

<table><tr><th>Argument</th><th>Description</th></tr><tr><td>file</td><td>(mandatory) compressed tsv file containing dosage sensitivity scores</td></tr><tr><td>cover</td><td>set value to 1 (0 by default) to report scores only if the variant covers the affected feature completely (e.g. - a CNV that duplicates the gene). The feature is a gene if using --database otherwise it is a transcript.</td></tr></table>
Usage examples:

<pre>mv DosageSensitivity.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin DosageSensitivity,file=/FULL_PATH_TO/Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz
./vep -i variations.vcf --plugin DosageSensitivity,file=/FULL_PATH_TO/Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz,cover=1</pre></td><td>Gene tolerance to change</td><td>-</td><td>Ensembl</td></tr>
<tr><td>Downstream</td><td>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.</td><td>Nearby features</td><td>-</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
	Usage examples: `mv Downstream.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Downstream`			
Draw	A VEP plugin that draws pictures of the transcript model showing the variant location. Takes 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. **Usage examples:** `mv Draw.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Draw`	Visualisation	• GD::Polygon • GD	Ensembl
Enformer	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds pre-calculated Enformer predictions of variant impact on chromatin and gene expression. The predictions have been aggregated across all 896 spatial bins to generate 5313 features corresponding to track prediction changes in differnet assays and cell types. This plugin is available for GRCh37 and GRCh38 Please cite the Enformer publication alongside the VEP if you use this resource: https://www.nature.com/articles/s41592-021-01252-x GRCh38 scores were lifted over using CrossMap from the Enformer scores available here - https://console.cloud.google.com/storage/browser/dm-enformer/variant-scores/1000-genomes/enformer Enformer scores can be downloaded from https://ftp.ensembl.org/pub/current_variation/Enformer for GRCh37 and GRCh38. The plugin can then be run as default to retrieve SAD (SNP Activity Difference (SAD) and SAR (Same as SAD, by computing np.log2(1 + model(alternate_sequence)) - np.log2(1 + model(reference_sequence)) scores from Enforme : `./vep -i variations.vcf --assembly GRCh38 --plugin Enformer,file=/path/to/Enformer/data.vcf.gz`	Regulatory impact	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	or run with option to only retrieve the SAD (SNP Activity Difference (SAD) scores - main variant effect score computed as model(alternate_sequence) - model(reference_sequence) score `./vep -i variations.vcf --assembly GRCh38 --plugin Enformer,file=/path/to/Enformer/data.vcf.gz,SAD=1` or run with option to only retrieve the SAR (Same as SAD, by computing np.log2(1 + model(alternate_sequence)) - np.log2(1 + model(reference_sequence)) score `./vep -i variations.vcf --assembly GRCh38 --plugin Enformer,file=/path/to/Enformer/data.vcf.gz,SAR=1` or run with option to also retrieve the principal component scores which are a reduced representation of a much bigger vector of the SAD and SAR after using principal component analysis (PCA) `./vep -i variations.vcf --assembly GRCh38 --plugin Enformer,file=/path/to/Enformer/data.vcf.gz,PC=1` The tabix utility must be installed in your path to use this plugin. Check https://github.com/samtools/htslib.git for instructions. **Usage examples:** `mv Enformer.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin Enformer,file=Enformer_grch38.vcf.gz`			
EVE	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds information from EVE (evolutionary model of variant effect). This plugin only report EVE scores for input variants and does not merge input lines to report on adjacent variants. It is only available for GRCh38. Please cite EVE publication alongside the VEP if you use this resource: https://www.nature.com/articles/s41586-021-04043-8 `##############################################` `# Bash script to merge all VCFs from EVE dataset. #` `##############################################` `### BEGIN` `# EVE input file can be downloaded from https://evemodel.org/api/proteins/bulk/download/` `# Input: VCF files by protein (vcf_files_missense_mutations inside zip folder)` `# Output: Compressed Merged VCF file (vcf.gz) + index file (.tbi)` `DATA_FOLDER=/<PATH-`	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer

```
TO>/vcf_files_missense_mutations # Fill this
line
OUTPUT_FOLDER=/<PATH-TO>/eve_plugin # Fill this
line
OUTPUT_NAME=eve_merged.vcf # Default output
name
# Get header from first VCF
cat `ls ${DATA_FOLDER}/*vcf | head -n1` >
header
# Get variants from all VCFs and add to a
single-file
ls ${DATA_FOLDER}/*vcf | while read VCF; do
grep -v '^#' ${VCF} >> variants; done
# Merge Header + Variants in a single file
cat header variants | \
awk '$1 ~ /^#/ {print $0;next} {print $0 |
"sort -k1,1V -k2,2n"}' >
${OUTPUT_FOLDER}/${OUTPUT_NAME};
# Remove temporary files
rm header variants
# Compress and index
bgzip ${OUTPUT_FOLDER}/${OUTPUT_NAME};
# If not installed, use: sudo apt install tabix
tabix ${OUTPUT_FOLDER}/${OUTPUT_NAME}.gz;
### END
```

Usage examples:

```
cp EVE.pm ${HOME}/.vep/Plugins
./vep -i variations.vcf --plugin
EVE,file=/path/to/eve/data.vcf.gz # By default,
Class75 is used.
./vep -i variations.vcf --plugin
EVE,file=/path/to/eve/data.vcf.gz,class_number=
60
```

FATHMM

A VEP plugin that gets FATHMM scores and predictions for missense variants.

Pathogenicity predictions

-

Ensembl

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/c
gi-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
```

Usage examples:

Plugin	Description	Category	External libraries	Developer
	`mv FATHMM.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin FATHMM,"python2 /path/to/fathmm/fathmm.py"`			
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. **Usage examples:** `mv FATHMM_MKL.pm ~/.vep/Plugins` `./vep -i input.vcf --plugin FATHMM_MKL,fathmm-MKL_Current.tab.gz`	Pathogenicity predictions	-	Ensembl
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 https://ftp.ebi.ac.uk/pub/databases/lrgex/list_LRGs_transcripts_xrefs.txt` **Usage examples:** `mv FlagLRG.pm ~/.vep/Plugins` `./vep -i variants.vcf --plugin FlagLRG,/path/to/list_LRGs_transcripts_xrefs.txt`	External ID	Text::CSV	Stephen Kazakoff
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. **Usage examples:** `mv FunMotifs.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin FunMotifs,/path/to/funmotifs/all_tissues.bed.gz,uterus` `./vep -i variations.vcf --plugin FunMotifs,/path/to/funmotifs/blood.funmotifs_sorted.bed.gz,fscore,dnase_seq` `Parameters Required:` `[0] : FunMotifs BED file` `[1]+ : List of columns to include within VEP`	Motif	-	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td><pre>output (e.g. fscore, skin, contactingdomain)</pre></td><td></td><td></td><td></td></tr>
<tr><td>G2P
gene2phenotype</td><td>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. This Plugin is available for GRCh38 and GRCh37.

For further information see: Thormann A, Halachev M, McLaren W, et al. Flexible and scalable diagnostic filtering of genomic variants using G2P with Ensembl VEP. Nature Communications. 2019 May;10(1):2373. doi:10.1038/s41467-019-10016-3. PMID: 31147538; PMCID: PMC6542828.

Options are passed to the plugin as key=value pairs, (defaults in parentheses):

<table><tr><th>Argument</th><th>Description</th></tr><tr><td>file</td><td>Path to G2P data file. The file needs to be uncompressed.
Download from https://www.ebi.ac.uk/gene2phenotype/downloads
Download from PanelApp</td></tr><tr><td>variant_include_list</td><td>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.</td></tr><tr><td>af_monoallelic</td><td>maximum allele frequency for inclusion for monoallelic genes (0.0001)</td></tr><tr><td>af_biallelic</td><td>maximum allele frequency for inclusion for biallelic genes (0.005)</td></tr><tr><td>confidence_levels</td><td>Confidence levels include: definitive, strong, moderate, limited Former confidence terms are still supported: confirmed, probable, possible, both RD and IF. Separate multiple values with &. https://www.ebi.ac.uk/gene2phenotype/terminology Default levels are confirmed and probable.</td></tr><tr><td>all_confidence_levels</td><td>Set to 1 to include all confidence levels Setting the value to 1 will overwrite any confidence levels provided with the confidence_levels option.</td></tr></table></td><td>Phenotype data and citations</td><td>List::Util qw(any)
Text::CSV
Scalar::Util qw(looks_like_number)
FileHandle
Cwd</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer

Argument	Description
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, gnomADe_r2.1.1, gnomADg, gnomADg_v3.1.2. Separate multiple values with &. VCF collections contain the following populations: • topmed - TOPMed (available for GRCh37 and GRCh38). • uk10k - ALSPAC, TWINSUK (available for GRCh37 and GRCh38). • gnomADe & gnomADe_r2.1.1 - gnomADe:AFR, gnomADe:ALL, gnomADe:AMR, gnomADe:ASJ, gnomADe:EAS, gnomADe:FIN, gnomADe:NFE, gnomADe:OTH, gnomADe:SAS (for GRCh37 and GRCh38 respectively). • gnomADg & gnomADg_v3.1.2 - gnomADg:AFR, gnomADg:ALL, gnomADg:AMR, gnomADg:ASJ, gnomADg:EAS, gnomADg:FIN, gnomADg:NFE, gnomADg:OTH (for GRCh37 and GRCh38 respectively). Need to use af_from_vcf parameter to use this option.
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

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>html_report</code></td><td>write all G2P complete genes and attributes to html file</td></tr>
<tr><td><code>filter_by_gene_symbol</code></td><td>set to 1 if filter by gene symbol. Do not set if filtering by HGNC_id. This option is set to 1 when using PanelApp files.</td></tr>
<tr><td><code>only_mane</code></td><td>set to 1 to ignore transcripts that are not MANE N/B - Information may be lost if this option is used.</td></tr>
</table>
For more information - https://www.ebi.ac.uk/gene2phenotype/g2p_vep_plugin
Example:
<pre>--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&gnomADe_r2.1.1
--plugin G2P,file=G2P.csv,af_from_vcf=1,af_from_vcf_keys=topmed&gnomADe_r2.1.1,confidence_levels='confirmed&probable&both RD and IF'
--plugin G2P,file=G2P.csv</pre>
<h3>Usage examples:</h3>
<pre>mv G2P.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin G2P,file=/path/to/G2P.csv</pre>
</td><td></td><td></td><td></td></tr>
<tr><td>GeneBe</td><td>A user-contributed VEP plugin that retrieves automatic ACMG variant classification data from https://genebe.net/

Please cite the GeneBe publication alongside the VEP if you use this resource: https://onlinelibrary.wiley.com/doi/10.1111/cge.14516 .

Please be advised that the GeneBe API is freely accessible for academic purposes only, with a limited number of queries per day, albeit at a high threshold. Kindly utilize this resource judiciously to ensure its availability for others. For further information, please visit https://genebe.net/about/api.

In order to extend your daily limits please make an account on https://genebe.net/ and use your username and API-key as follows:

<pre>./vep -i variations.vcf --plugin GeneBe,user=example@email.com,password=your_api</pre></td><td>Variant data</td><td>JSON</td><td>• Ensembl
• Piotr Stawinski</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
	`_key` **Usage examples:** `mv GeneBe.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin GeneBe`			
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. You will need to download the GeneSplicer binary and data from ftp://ftp.ccb.jhu.edu/pub/software/genesplicer/GeneSplicer.tar.gz. Extract the folder using: `tar -xzf GeneSplicer.tar.gz` GeneSplicer comes with precompiled binaries for multiple systems. If the provided binaries do not run, compile `genesplicer` from source: `cd $GS/sources` `# if macOS, run this step` `[[ $(uname -s) == "Darwin" ]] && perl -pi -e "s/^main  /int main  /" genesplicer.cpp` `make` `cd -` `./vep [options] --plugin GeneSplicer,$GS/sources/genesplicer,$GS/human` Predicted splicing regions that overlap the variant are reported in the output with a `/`-separated string (e.g., `loss/acceptor/727006-727007/High/16.231924`) consisting of the following data by this order: 1. state (no_change, diff, gain, loss) 2. type (donor, acceptor) 3. coordinates (start-end) 4. confidence (Low, Medium, High) 5. score 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, such as `diff/donor/621915-621914/Medium-Medium/7.020731-6.988368`. Several key=value parameters can be modified in the the plugin string: **Argument** — **Description** `training` — (mandatory) directory to species-specific training data, such as `GeneSplicer/human`	Splicing predictions	Digest::MD5 qw(md5_hex)	Ensembl

Plugin	Description	Category	External libraries	Developer

Argument	Description
`binary`	path to `genesplicer` binary (default: `genesplicer`)
`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,binary=$GS/bin/linux/genesplicer,training=$GS/human,context=200,tmpdir=/mytmp
```

When using VEP Docker/Singularity, the `binary` argument can be ommitted, as the `genesplicer` command is exported in the $PATH variable and is thus automatically detected by the plugin:

```
--plugin
GeneSplicer,training=$GS/human,context=200,tmpdir=/mytmp
```

Usage examples:

```
mv GeneSplicer.pm ~/.vep/Plugins
./vep -i variants.vcf --plugin
GeneSplicer,binary=$GS/bin/linux/genesplicer,training=$GS/human
./vep -i variants.vcf --plugin
GeneSplicer,binary=$GS/bin/linux/genesplicer,training=$GS/human,context=200,tmpdir=/mytmp

# VEP Docker/Singularity: if 'genesplicer' is a command available in $PATH,
# there is no need to specify the location of the binary
./vep -i variants.vcf --plugin
GeneSplicer,training=$GS/human
```

Geno2MP — A VEP plugin that adds information from Geno2MP, a web-accessible database of rare variant genotypes linked to phenotypic information.

Category: Phenotype data and citations

External libraries: -

Developer: Ensembl

Parameters can be set using a key=value system:

Argument	Description
`file`	VCF file containing Geno2MP data
`cols`	colon-delimited list of Geno2MP columns to return from INFO fields (by default it only returns the column HPO_CT)
`url`	build and return URL to Geno2MP variant page (boolean; 0 by default); the variant location in Geno2MP website is based on GRCh37 coordinates

Plugin	Description	Category	External libraries	Developer
	Please cite Geno2MP alongside the VEP if you use this resource: Geno2MP, NHGRI/NHLBI University of Washington-Center for Mendelian Genomics (UW-CMG), Seattle, WA (URL: http://geno2mp.gs.washington.edu [date (month, yr) accessed]). **Usage examples:** `cp Geno2MP.pm ${HOME}/.vep/Plugins` `./vep -i variations.vcf --plugin Geno2MP,file=/path/to/Geno2MP/data.vcf.gz` `# Return more columns from Geno2MP VCF file` `./vep -i variations.vcf --plugin Geno2MP,file=/path/to/Geno2MP/data.vcf.gz,cols=HPO_CT:FXN:nhomalt_male_aff:nhomalt_male_unaff` `# Build and return Geno2MP URL based on GRCh37 variant location` `./vep -i variations.vcf --plugin Geno2MP,file=/path/to/Geno2MP/data.vcf.gz,url=1`			
gnomADc	A VEP plugin that retrieves gnomAD annotation from either the genome or exome coverage files, available here: https://gnomad.broadinstitute.org/downloads To download the gnomad coverage file in TSV format: for Assembly GRCh37: gnomad genomes: `wget https://storage.googleapis.com/gcp-public-data--gnomad/release/2.1/coverage/genomes/gnomad.genomes.coverage.summary.tsv.bgz --no-check-certificate` gnomad exomes: `wget https://storage.googleapis.com/gcp-public-data--gnomad/release/2.1/coverage/exomes/gnomad.exomes.coverage.summary.tsv.bgz --no-check-certificate` for Assembly GRCh38: gnomad genomes: `wget https://storage.googleapis.com/gcp-public-data--gnomad/release/3.0.1/coverage/genomes/gnomad.genomes.r3.0.1.coverage.summary.tsv.bgz --no-check-certificate` Necessary before using the plugin for Assembly GRCh37: The following steps are necessary to tabix the gnomad genomes coverage file : `gunzip -c gnomad.genomes.coverage.summary.tsv.bgz | sed '1s/.*/#&/' > gnomad.genomes.tabbed.tsv` `bgzip gnomad.genomes.tabbed.tsv` `tabix -s 1 -b 2 -e 2 gnomad.genomes.tabbed.tsv.gz`	Frequency data	• File::Spec • File::Basename	Stephen Kazakoff

Plugin	Description	Category	External libraries	Developer
	The following steps are neccessary to tabix the gnomad exomes coverage file : `gunzip -c gnomad.exomes.coverage.summary.tsv.bgz | sed '1s/.*/#&/' > gnomad.exomes.tabbed.tsv` `bgzip gnomad.exomes.tabbed.tsv` `tabix -s 1 -b 2 -e 2 gnomad.exomes.tabbed.tsv.gz` for Assembly GRCh38: The following steps are necessary to tabix the gnomad genomes coverage file : `gunzip -c gnomad.genomes.r3.0.1.coverage.summary.tsv.bgz | sed '1s/.*/#&/' > gnomad.genomesv3.tabbed.tsv` `sed "1s/locus/chr\tpos/; s/:/\t/g" gnomad.genomesv3.tabbed.tsv > gnomad.ch.genomesv3.tabbed.tsv` `bgzip gnomad.ch.genomesv3.tabbed.tsv` `tabix -s 1 -b 2 -e 2 gnomad.ch.genomesv3.tabbed.tsv` This plugin also tries to be backwards compatible with older versions of the coverage summary files, including releases 2.0.1 and 2.0.2. These releases provide one coverage file per chromosome and these can be used "as-is" without requiring any preprocessing. If you use this plugin, please see the terms and data information: https://gnomad.broadinstitute.org/terms You must have the Bio::DB::HTS module or the tabix utility must be installed in your path to use this plugin. **Usage examples:** `mv gnomADc.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin gnomADc,/path/to/gnomad.tsv.gz`			
GO Gene Ontology	A VEP plugin that retrieves Gene Ontology (GO) terms associated with transcripts (e.g. GRCh38) or their translations (e.g. GRCh37) using custom GFF annotation containing GO terms. The custom GFF files are automatically created if the input file do not exist by querying the Ensembl core database, according to database version, species and assembly used in VEP. Note that automatic retrieval fails if using the --offline option. The GFF files containing the GO terms are saved to and loaded from the working directory by default. To change this, provide a directory path as an argument: `--plugin GO,dir=${HOME}/go_terms` If your GFF file has a custom name, please provide the filename directly: `--plugin GO,file=${HOME}/custom_go_terms.gff.gz` The GO terms can also be fetched by gene match (either gene Ensembl ID or gene symbol) instead:	Phenotype data and citations	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`` --plugin GO,match=gene --plugin GO,match=gene_symbol ``` To create/use a custom GFF file, these programs must be installed in your path: - The GNU zgrep and GNU sort commands to create the GFF file. - The tabix and bgzip utilities to create and read the GFF file: check https://github.com/samtools/htslib.git for installation instructions. Alternatively, for compatibility purposes, the plugin allows to use a remote connection to the Ensembl API by using "remote" as a parameter. This method retrieves GO terms one by one at both the transcript and translation level. This is not compatible with any other parameters: ``` --plugin GO,remote ``` **Usage examples:** ``` mv GO.pm ~/.vep/Plugins # automatically fetch GFF files with GO terms and annotate input with GO terms # not compatible with --offline option ./vep -i variations.vcf --plugin GO # set directory used to write and read GFF files with GO terms ./vep -i variations.vcf --plugin GO,dir=${HOME}/go_terms # annotate input with GO terms from custom GFF file ./vep -i variations.vcf --plugin GO,file=${HOME}/custom_go_terms.gff.gz # annotate input based on gene identifiers instead of transcripts/translations ./vep -i variations.vcf --plugin GO,match=gene # use remote connection (available for compatibility purposes) ./vep -i variations.vcf --plugin GO,remote ```			
GWAS	A VEP plugin that retrieves relevant NHGRI-EBI GWAS Catalog data given the file. This plugin supports both the curated data that is found in the download section of the NHGRI-EBI GWAS Catalog website and the summary statistics file. By default the plugin assumes the file provided is the curated file but you can pass "type=sstate" to say you want to annotate with a summary statistics file. Please cite the following publication alongside the VEP if you use this resource: https://pubmed.ncbi.nlm.nih.gov/30445434/ Pre-requisites: For curated NHGRI-EBI GWAS Catalog file - GWAS files can be downloaded from - https://www.ebi.ac.uk/gwas/api/search/downloads/alternative	Phenotype data and citations	- Storable qw(dclone) - File::Basename	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>
For summary statistics file - The plugin can process the harmonised version of the summary statistics file. Which can be downloaded from the FTP site - http://ftp.ebi.ac.uk/pub/databases/gwas/summary_statistics

They are under directory with related to their specific GCST id. For example - http://ftp.ebi.ac.uk/pub/databases/gwas/summary_statistics/GCST000001-GCST001000/GCST000028/harmonised/17463246-GCST000028-EFO_0001360.h.tsv.gz

Please keep the filename format as it is because filename is parsed to get information.

When run for the first time for either type of file, the plugin will create a processed file that have genomic locations and indexed and put it under the --dir location determined by Ensembl VEP. If db=1 option is used, depending on the file size it might take hour(s) to create the processed file. Subsequent runs will be faster as the plugin will be using the already generated processed file. This option is not used by default and the variant information is generally taken directly from the file provided.

Options are passed to the plugin as key=value pairs:
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>file</td><td>(mandatory) Path to GWAS curated or summary statistics file</td></tr>
<tr><td>type</td><td>type of the file. Valid values are "curated" and "sstate" (summary statistics). Default is "curated".</td></tr>
<tr><td>verbose</td><td>display info level messages. Valid values are 0 or 1. Default is 0.</td></tr>
<tr><td>db</td><td>get variant information from Ensembl database during creation of processed file. Valid values are 0 or 1. Default is 0 (variant information is retrieved from curated file)</td></tr>
</table>
<h3>Usage examples:</h3>
<pre>
mv GWAS.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
GWAS,file=/FULL_PATH_TO/gwas_catalog_v1.0.2-associations_e107_r2022-09-14.tsv
./vep -i variations.vcf --plugin
GWAS,type=sstate,file=/FULL_PATH_TO/17463246-GCST000028-EFO_0001360.h.tsv.gz
</pre>
</td><td></td><td></td><td></td></tr>
<tr><td>HGVSIntronOffset</td><td>
A VEP plugin for the Ensembl Variant Effect Predictor (VEP) that returns HGVS intron start and end offsets. To be used with --hgvs option.

<h3>Usage examples:</h3>
<pre>
mv HGVSIntronOffset.pm ~/.vep/Plugins
./vep -i variants.vcf --hgvs --plugin
HGVSIntronOffset
</pre>
</td><td>HGVS</td><td>-</td><td>Stephen Kazakoff</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
IntAct	A VEP plugin that retrieves molecular interaction data for variants as reprted by IntAct database. Please cite the IntAct publication alongside the VEP if you use this resource: https://pubmed.ncbi.nlm.nih.gov/24234451/ Pre-requisites: 1. IntAct files can be downloaded from - https://ftp.ebi.ac.uk/pub/databases/intact/current/various 2. The genomic location mapped file needs to be tabix indexed. You can do this by following commands - a) filter, sort and then zip `grep -v -e '^$' -e '^[#\-]' mutation_gc_map.txt | sed '1s/.*/#&/' | awk -F "\t" 'BEGIN { OFS="\t"} {if ($2 > $3) {a=$2; $2=$3; $3=a}; print $0 }' | sort -k1,1 -k2,2n -k3,3n | bgzip > mutation_gc_map.txt.gz` b) create tabix indexed file - `tabix -s 1 -b 2 -e 3 -f mutation_gc_map.txt.gz` 3. As you have already noticed, tabix utility must be installed in your path to use this plugin. Options are passed to the plugin as key=value pairs: **Argument** — **Description** `mapping_file` — (mandatory) Path to tabix-indexed genomic location mapped file `mutation_file` — (mandatory) Path to IntAct data file By default the output will always contain feature_type and interaction_ac from the IntAct data file. You can also add more fields using the following key=value options - **Argument** — **Description** `feature_ac` — Set value to 1 to include Feature AC in the output `feature_short_label` — Set value to 1 to include Feature short label in the output `feature_annotation` — Set value to 1 to include Feature annotation in the output `ap_ac` — Set value to 1 to include Affected protein AC in the output `interaction_participants` — Set value to 1 to include Interaction participants in the output `pmid` — Set value to 1 to include PubMedID in the output	Functional effect	-	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>There are also two other key=value options for customizing the output -

<table><tr><th>Argument</th><th>Description</th></tr><tr><td>all</td><td>Set value to 1 to include all the fields</td></tr><tr><td>minimal</td><td>Set value to 1 to overwrite default behavior and include only interaction_ac in the output by default</td></tr></table>
See what these options mean - https://www.ebi.ac.uk/intact/download/datasets#mutations

Note that, interaction accession can be used to link to full details on the interaction website. For example, where the VEP output reports an interaction_ac of EBI-12501485, the URL would be :

https://www.ebi.ac.uk/intact/details/interaction/EBI-12501485

Usage examples:

<pre>mv IntAct.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
IntAct,mutation_file=/FULL_PATH_TO_IntAct_FILE/
mutations.tsv,mapping_file=/FULL_PATH_TO_IntAct
_FILE/mutation_gc_map.txt.gz
./vep -i variations.vcf --plugin
IntAct,mutation_file=/FULL_PATH_TO_IntAct_FILE/
mutations.tsv,mapping_file=/FULL_PATH_TO_IntAct
_FILE/mutation_gc_map.txt.gz,minimal=1</pre></td><td></td><td></td><td></td></tr>
<tr><td>LD
Linkage Disequilibrium</td><td>A VEP plugin 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: 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:

<pre>./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</pre>
Usage examples:

<pre>mv LD.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
LD,1000GENOMES:phase_3:CEU,0.8
./vep -i variations.vcf --plugin</pre></td><td>Variant data</td><td>-</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
	LD,'populations=1000GENOMES:phase_3:CEU&1000GENOMES:phase_3:PUR&1000GENOMES:phase_3:STU',0.8			
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. **Usage examples:** `mv LocalID.pm ~/.vep/Plugins` `## first run create database` `# EITHER create from Ensembl variation database` `# VERY slow but includes variant synonyms, if not required see next command` `./vep -i variant_ids.txt --plugin LocalID,create_db=1 -safe` `# OR create from cache directory` `# faster but does not include synonyms` `# parameter passed to from_cache may be full path to cache e.g. $HOME/.vep/homo_sapiens/88_GRCh38` `# cache may be tabix converted or in default state (http://www.ensembl.org/info/docs/tools/vep/script/vep_cache.html#convert)` `./vep -i variant_ids.txt --plugin LocalID,create_db=1,from_cache=1 -safe` `# subsequent runs` `./vep -i variant_ids.txt --plugin LocalID` `# db file can be specified with db=[file]` `# default file name is $HOME/.vep/[species]_[version]_[assembly].variant_ids.sqlite3` `./vep -i variant_ids.txt --plugin LocalID,db=my_db_file.txt`	Look up	-	Ensembl
LOEUF	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds the LOEUF scores to VEP output. LOEUF stands for the "loss-of-function observed/expected upper bound fraction." The score can be added matching by either transcript or gene. When matched by gene: If multiple transcripts are available for a gene, the most severe score is reported. NB: The plugin currently does not add the score for downstream_gene_variant and upstream_gene_variant Please cite the LOEUF publication alongside the VEP if you use this resource: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7334197/ LOEUF scores can be downloaded from GRCh37: https://gnomad.broadinstitute.org/downloads#v2-constraint (pLoF Metrics by Gene TSV) GRCh38:	Gene tolerance to change	Scalar::Util qw(looks_like_number)	Ensembl

Plugin	Description	Category	External libraries	Developer
	https://gnomad.broadinstitute.org/downloads#v4-constraint (Constraint metrics TSV) For GRCh37: These files can be tabix-processed by: `zcat gnomad.v2.1.1.lof_metrics.by_gene.txt.bgz | (head -n 1 && tail -n +2 | sort -t$'\t' -k 76,76 -k 77,77n ) > loeuf_temp.tsv` `sed '1s/.*/#&/' loeuf_temp.tsv > loeuf_dataset.tsv` `bgzip loeuf_dataset.tsv` `tabix -f -s 76 -b 77 -e 78 loeuf_dataset.tsv.gz` For GRCh38: The GRCh38 file does not have gene co-ordinates information. First you need to add the gene co-ordiates information. You can use the Ensembl Perl API to create a script and perform that - https://www.ensembl.org/info/docs/api/core/index.html. After adding the start and end position of the genes at the last 2 columns you can process the file as follows: `cat gnomad.v4.1.constraint_metrics_with_coordinates.tsv \| (head -n 1 && tail -n +2 \| sort -t$'\t' -k 53,53 -k 56,56n ) > loeuf_grch38_temp.tsv` `sed '1s/.*/#&/' loeuf_grch38_temp.tsv > loeuf_dataset_grch38.tsv` `bgzip loeuf_dataset_grch38.tsv` `tabix -f -s 53 -b 56 -e 57 loeuf_dataset_grch38.tsv.gz` The tabix utility must be installed in your path to use this plugin. **Usage examples:** `mv LOEUF.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin LOEUF,file=/path/to/loeuf/data.tsv.gz,match_by=gene` `./vep -i variations.vcf --plugin LOEUF,file=/path/to/loeuf/data.tsv.gz,match_by=transcript`			
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`	Pathogenicity predictions	DBI	Ensembl

Plugin	Description	Category	External libraries	Developer
	Usage examples: `mv LoFtool.pm ~/.vep/Plugins` `mv LoFtool_scores.txt ~/.vep/Plugins` `./vep -i variants.vcf --plugin LoFtool`			
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. **Usage examples:** `mv LOVD.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin LOVD`	Variant data	LWP::UserAgent	Ensembl
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. Please cite the Mastermind publication alongside the VEP if you use this resource: https://www.frontiersin.org/article/10.3389/fgene.2020.577152 Running options: The plugin has multiple parameters, the first one is expected to be the file name path which can be followed by 3 optional flags. Default: the plugin matches the citation data with the specific mutation. Using first flag `1`: returns the citations for all mutations/transcripts. Using the second flag `1`: only returns the Mastermind variant identifier(s). Using the third flag `1`: also returns the Mastermind URL. 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. Link to the Genomenon Mastermind Genomic Search Engine; To build the URL, substitute the `gene:key` in the following link with the value from MMID3: https://mastermind.genomenon.com/detail?mutation=gene:key If the third flag is used then the built URL is returned and it's identified by `URL`. 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:	Phenotype data and citations	-	Ensembl

Plugin	Description	Category	External libraries	Developer

```
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:

```
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/mastermind_cited_varia
nts_reference-XXXX.XX.XX.GRChXX-vcf.gz
```

or with an option to not filter by mutations (first flag):

```
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/mastermind_cited_varia
nts_reference-XXXX.XX.XX.GRChXX-
vcf.gz,mutations=1
```

or with an option to only return `MMID3` e.g. the Mastermind variant identifier as gene:key (second flag):

```
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/mastermind_cited_varia
nts_reference-XXXX.XX.XX.GRChXX-
vcf.gz,mutations=0,var_iden=1
```

or with an option to also return the Mastermind URL (third flag):

```
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/mastermind_cited_varia
nts_reference-XXXX.XX.XX.GRChXX-
vcf.gz,mutations=0,var_iden=0,url=1
```

Note: when running VEP in offline mode Mastermind requires a fasta file (--fasta)

Usage examples:

```
mv Mastermind.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/data.vcf.gz
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/data.vcf.gz,mutations=
1
./vep -i variations.vcf --plugin
Mastermind,file=/path/to/data.vcf.gz,mutations=
0,var_iden=1
./vep -i variations.vcf --plugin
```

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td><pre>Mastermind,file=/path/to/data.vcf.gz,mutations=0,var_iden=0,url=1</pre></td><td></td><td></td><td></td></tr>
<tr><td>MaveDB</td><td>A VEP plugin that retrieves data from MaveDB (https://www.mavedb.org), a database that contains multiplex assays of variant effect, including deep mutational scans and massively parallel report assays.

To run the MaveDB plugin, please download the following files containing MaveDB data for GRCh38 (we do not currently host data for other assemblies):

• https://ftp.ensembl.org/pub/current_variation/MaveDB/MaveDB_variants.tsv.gz
• https://ftp.ensembl.org/pub/current_variation/MaveDB/MaveDB_variants.tsv.gz.tbi

Options are passed to the plugin as key=value pairs:

<table><tr><th>Argument</th><th>Description</th></tr><tr><td>file</td><td>(mandatory) Tabix-indexed MaveDB file</td></tr><tr><td>cols</td><td>Colon-separated columns to print from MaveDB files; if set to all, all columns are printed (default: urn:score:nt:pro)</td></tr><tr><td>single_aminoacid_changes</td><td>Return matches for single aminoacid changes only; if disabled, return all matches associated with a genetic variant (default: 1)</td></tr><tr><td>transcript_match</td><td>Return results only if (Ensembl or RefSeq) transcript identifiers match (default: 1)</td></tr></table>
Please cite the MaveDB publication alongside the VEP if you use this resource: https://doi.org/10.1186/s13059-019-1845-6

The tabix utility must be installed in your path to use this plugin.

Usage examples:

<pre>mv MaveDB.pm ~/.vep/Plugins

print only scores for single aminoacid changes from MaveDB data (default)
./vep -i variations.vcf --plugin MaveDB,file=/full/path/to/data.csv.gz

print all scores associated with the genetic variant
./vep -i variations.vcf --plugin MaveDB,file=/full/path/to/data.csv.gz,single_aminoacid_changes=0

print all columns from MaveDB data
./vep -i variations.vcf --plugin MaveDB,file=/full/path/to/data.csv.gz,cols=all</pre></td><td>Functional effect</td><td>• Bio::SeqUtils
• File::Basename</td><td>Ensembl</td></tr>
<tr><td>MaxEntScan</td><td>This is a plugin for the Ensembl Variant Effect Predictor (VEP) that runs MaxEntScan (http://hollywood.mit.edu/burgelab/maxent/Xmaxentscan_scoreseq.html) to get splice site predictions.</td><td>Splicing predictions</td><td>Digest::MD5 qw(md5_hex)</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
	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://hollywood.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. **Usage examples:** `mv MaxEntScan.pm ~/.vep/Plugins` `./vep -i variants.vcf --plugin MaxEntScan,/path/to/maxentscan/fordownload` `./vep -i variants.vcf --plugin MaxEntScan,/path/to/maxentscan/fordownload,SWA,NCSS`			
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: https://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. **Usage examples:**	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`` mv MPC.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin MPC,fordist_constraint_official_mpc_values.txt. gz ```			
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 http://biosig.unimelb.edu.au/mtr-viewer/downloads The following steps are necessary before running the plugin ``` gzip -d mtrflatfile_2.0.txt.gz # to unzip the text file cat mtrflatfile_2.0.txt | tr " " "\t" > mtrflatfile_2.00.tsv # to change the file to a tabbed delimited file sed '1s/.*/#&/' mtrflatfile_2.00.tsv > mtrflatfile_2.0.tsv # to add # to the first line of the file bgzip mtrflatfile_2.0.tsv tabix -f -s 1 -b 2 -e 2 mtrflatfile_2.0.tsv.gz ``` NB: Data are available for GRCh37 only **Usage examples:** ``` mv MTR.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin MTR,mtrflatfile_2.0.tsv.gz ```	Pathogenicity predictions	-	• Slave Petrovski • Michael Silk
mutfunc	A VEP plugin that retrieves data from mutfunc db predicting destabilization of protein structure, interaction interface, and motif. Please cite the mutfunc publication alongside the VEP if you use this resource: http://msb.embopress.org/content/14/12/e8430 Pre-requisites: 1. The data file. mutfunc SQLite db can be downloaded from - https://ftp.ensembl.org/pub/current_variation/mutfunc/mutfunc_data.db 2. If you are using --offline please provide a FASTA file as this plugin requires the translation sequence to function. Options are passed to the plugin as key=value pairs: Argument: db — Description: (mandatory) Path to SQLite database containing data for other analysis.	Protein annotation	• List::MoreUtils qw(first_index) • Compress::Zlib • Digest::MD5 qw(md5_hex) • DBI	Ensembl

Plugin	Description	Category	External libraries	Developer
	<table><tr><th>Argument</th><th>Description</th></tr><tr><td>motif</td><td>Select this option to have mutfunc motif analysis in the output</td></tr><tr><td>int</td><td>Select this option to have mutfunc protein interection analysis in the output</td></tr><tr><td>mod</td><td>Select this option to have mutfunc protein structure analysis in the output</td></tr><tr><td>exp</td><td>Select this option to have mutfunc protein structure (experimental) analysis in the output</td></tr><tr><td>extended</td><td>By default mutfunc outputs the most significant field for any analysis. Select this option to get more verbose output.</td></tr></table> By default all of the four type of analysis (motif, int, mod, and exp) data are available in the output. But if you want to have some selected analysis and not all of them just select the relevant options. **Usage examples:** ``` mv mutfunc.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin mutfunc,motif=1,extended=1,db=/FULL_PATH_TO/mutfunc_data.db ./vep -i variations.vcf --plugin mutfunc,db=/FULL_PATH_TO/mutfunc_data.db ```			
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 key=value parameters can be altered by passing them to the plugin command: <table><tr><th>Argument</th><th>Description</th></tr><tr><td>max_range</td><td>maximum search range in bp (default: 10000)</td></tr></table> Parameters are passed e.g.: ``` --plugin NearestExonJB,max_range=50000 ``` **Usage examples:** ``` mv NearestExonJB.pm ~/.vep/Plugins ./vep -i variations.vcf --cache --plugin NearestExonJB ```	Nearby features	-	Ensembl
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 key=value parameters can be altered by passing them to the plugin command:	Nearby features	-	Ensembl

Plugin	Description	Category	External libraries	Developer

Argument	Description
`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.

Usage examples:

```
mv NearestGene.pm ~/.vep/Plugins
./vep -i variations.vcf --cache --plugin
NearestGene
```

neXtProt

This is a plugin for the Ensembl Variant Effect Predictor (VEP) that retrieves data for missense and stop gain variants from neXtProt, which is a comprehensive human-centric discovery platform that offers integration of and navigation through protein-related data for example, variant information, localization and interactions (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 is run for each variant.

The neXtProt_headers.txt file is a requirement for running this plugin and is found alongside the plugin in the VEP_plugins GitHub repository. The file contains the RDF entities extracted from https://snorql.nextprot.org/

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:

Argument	Description
`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 with different data separated by `&`. Use file `neXtProt_headers.txt` to check which data (labels) are available. Example: --plugin neXtProt,return_values=`Domain&InteractingRegion`
`url`	Set value to 1 to include the URL to link to the neXtProt entry.
`all_labe`	Set value to 1 to include all labels, even if data is not available.

Protein data

JSON::XS

Ensembl

Plugin	Description	Category	External libraries	Developer
	<table><tr><th>Argument</th><th>Description</th></tr><tr><td>ls</td><td></td></tr><tr><td>position</td><td>Set value to 1 to include the start and end position in the protein.</td></tr></table> (*) 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=-; Of notice, multiple values can be returned for the same label. In this case, the values will be separeted by | for tab and txt format, and & for VCF format. N/B: This plugin requires a connection to the Ensembl database, and can not be used in offline mode. 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 **Usage examples:** mv neXtProt.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin neXtProt ./vep -i variations.vcf --plugin neXtProt,max_set=1			
NMD	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that predicts if a variant allows the transcript escape nonsense-mediated mRNA decay based on certain rules. The rules are : 1. The variant location falls in the last exon of the transcript. vvvv ES...EE..I.ES...EE.I.ES....EE.I.ES....EE	Transcript annotation	-	Ensembl

Plugin	Description	Category	External libraries	Developer

(ES= exon_start,EE = exon_end, I = intron, v = variant location)

2. The variant location falls 50 bases upstream of the penultimate (second to the last) exon.

```
                                  vvv

ES...EE..I.ES...EE.I.ES....EE.I.ES....EE
```

(ES= exon_start,EE = exon_end, I = intron, v = variant location)

3. The variant falls in the first 100 coding bases in the transcript.

```
          vvv

..ES...EE..I.ES...EE.I.ES....EE.I.ES....EE
```

(ES= exon_start,EE = exon_end, I = intron, v = variant location)

4. If the variant is in an intronless transcript, meaning only one exon exist in the transcript.

The additional term NMD-escaping variant (nonsense-mediated mRNA decay escaping variants) will be added if the variant matches any of the rules.

REFERENCES :

- Identifying Genes Whose Mutant Transcripts Cause Dominant Disease Traits by Potential Gain-of-Function Alleles (Coban-Akdemir, 2018)
- The rules and impact of nonsense-mediated mRNA decay in human cancers (Lindeboom, 2016)

Usage examples:

```
mv NMD.pm  ~/.vep/Plugins
./vep -i variations.vcf --plugin NMD
```

OpenTargets

A VEP plugin that integrates data from Open Targets Genetics (https://genetics.opentargets.org), a tool that highlights variant-centric statistical evidence to allow both prioritisation of candidate causal variants at trait-associated loci and identification of potential drug targets.

Data from Open Targets Genetics includes locus-to-gene (L2G) scores to predict causal genes at GWAS loci.

The tabix utility must be installed in your path to use this plugin. The Open Targets Genetics file and respective index (TBI) file can be downloaded from: https://ftp.ebi.ac.uk/pub/databases/opentargets/genetics/latest/OTGenetics_VEP

Options are passed to the plugin as key=value pairs:

Argument	Description
`file`	(mandatory) Tabix-indexed file from Open Targets Genetics
`cols`	(optional) Colon-separated list of columns to return from the plugin file (default: "l2g:geneId"); use `all` to print all data

Variant data

- Bio::SeqUtils
- File::Basename

Ensembl

Plugin	Description	Category	External libraries	Developer
	Please cite the Open Targets Genetics publication alongside the VEP if you use this resource: https://doi.org/10.1093/nar/gkaa84 **Usage examples:** `mv OpenTargets.pm ~/.vep/Plugins` `# print Open Targets Genetics scores and respective gene identifiers (default)` `./vep -i variations.vcf --plugin OpenTargets,file=path/to/data.tsv.bz` `# print all information from Open Targets Genetics` `./vep -i variations.vcf --plugin OpenTargets,file=path/to/data.tsv.bz,cols=all`			
Paralogues	A VEP plugin that fetches variants overlapping the genomic coordinates of amino acids aligned between paralogue proteins. This is useful to predict the pathogenicity of variants in paralogue positions. This plugin can determine paralogue regions for a variant based on: 1. Pre-computed matches between genomic regions and paralogue variants. For this approach, either download the file calculated using ClinVar variants and respective TBI from https://ftp.ensembl.org/pub/current_variation/Paralogues or create such matches file yourself. Details on how to create such `matches` file can be found below. 2. Ensembl paralogue annotation. These versatile annotations can look up paralogue regions for all variants from any species with Ensembl paralogues, but take longer to process. After retrieving the paralogue regions, this plugin fetches variants overlapping those regions from one of the following sources (by this order): 1. Custom VCF via the `vcf` parameter 2. VEP cache (in cache/offline mode) 3. Ensembl API (in database mode) To create a `matches` file based on a custom set of variants, run VEP using `--plugin Paralogues,regions=1,min_perc_cov=0,min_perc_pos=0,clnsig=ignore` and the `--vcf` option. Afterwards, process the output of the VEP command: `perl -e "use Paralogues; Paralogues::prepare_matches_file(variant_effect_output.txt)"` Options are passed to the plugin as key=value pairs: **Argument** — **Description** `matches` — Tabix-indexed TSV file with pre-computed matches between genomic regions and paralogue variants (fastest method); this option is incompatible with the `paralogues` and `vcf` options `dir` — Directory with paralogue annotation (the annotation is created in this folder if the paralogue annotation files do not exist)	Variant data	• Compress::Zlib • Bio::SimpleAlign • File::Spec • List::Util qw(any) • File::Basename	Ensembl

Plugin	Description	Category	External libraries	Developer

Argument	Description
paralogues	Tabix-indexed TSV file with paralogue annotation (if the file does not exist, the annotation is automatically created); if set to `remote`, the annotation is fetched but not stored
vcf	Tabix-indexed VCF file to fetch variant information (if not used, variants are fetched from VEP cache in cache/offline mode or Ensembl API in database mode)
fields	Colon-separated list of information from paralogue variants to output (default: `identifier:alleles:clinical_significance`); keyword `all` can be used to print all fields; available fields include `identifier`, `chromosome`, `start`, `alleles`, `perc_cov`, `perc_pos`, and `clinical_significance` (if `clnsig_col` is defined for custom VCF); additional fields are available depending on variant source: • VEP cache: `end` and `strand` • Ensembl API: `end`, `strand`, `source`, `consequence` and `gene_symbol` • Custom VCF: `quality`, `filter` and name of INFO fields • Matches file: check column names in file header
clnsig	Clinical significance term to filter variants (default: `pathogenic`); use `ignore` to fetch all paralogue variants, regardless of clinical significance
clnsig_match	Type of match when filtering variants based on option `clnsig`: `partial` (default), `exact` or `regex`
clnsig_col	Column name containing clinical significance in custom VCF (required with `vcf` option and if `clnsig` is not `ignore`)
min_perc_cov	Minimum alignment percentage of the peptide associated with the input variant (default: 0)
min_perc_pos	Minimum percentage of positivity (similarity) between both homologues (default: 50)

Plugin	Description	Category	External libraries	Developer

Argument	Description
regions	Boolean value to return regions used to look up paralogue variants (default: 1)

The tabix utility must be installed in your path to read the paralogue annotation, the custom VCF file and the matches file.

Usage examples:

```
mv Paralogues.pm ~/.vep/Plugins

# Find paralogue regions of all input variants
using Ensembl paralogue annotation
# (automatically created if not in current
directory) and fetch variants within
# those regions from VEP cache and whose
clinical significance partially
# matches 'pathogenic'
./vep -i variations.vcf --cache --plugin
Paralogues

# Find paralogue regions of input variants
using Ensembl paralogue annotation
# (automatically created if not in current
directory) and fetch variants within
# those regions from a custom VCF file
(regardless of their clinical significance)
./vep -i variations.vcf --cache --plugin
Paralogues,vcf=/path/to/file.vcf,clnsig=ignore

# Same using a custom VCF file but filtering
for 'pathogenic' variants
./vep -i variations.vcf --cache --plugin
Paralogues,vcf=/path/to/file.vcf,clnsig_col=CLN
SIG

# Same but output different fields
./vep -i variations.vcf --cache --plugin
Paralogues,vcf=/path/to/file.vcf.gz,clnsig_col=
CLNSIG,fields=identifier:alleles:CLNSIG:CLNVI:G
ENEINFO

# Use a file with regions matched to paralogue
variants -- fastest method;
# download 'matches' files from
https://ftp.ensembl.org/pub/current_variation/P
aralogues
./vep -i variations.vcf --cache --plugin
Paralogues,matches=Paralogues.pm_homo_sapiens_1
13_GRCh38_clinvar_20240107.tsv.gz,clnsig=ignore

# Same using a 'matches' file but filtering for
'pathogenic' variants (default)
./vep -i variations.vcf --cache --plugin
Paralogues,matches=Paralogues.pm_homo_sapiens_1
13_GRCh38_clinvar_20240107.tsv.gz

# Fetch all Ensembl variants in paralogue
proteins using only the Ensembl API
# (requires database access)
```

Plugin	Description	Category	External libraries	Developer
	`./vep -i variations.vcf --database --plugin Paralogues,mode=remote,clnsig=ignore`			
PhenotypeOrthologous	A VEP plugin that retrieves phenotype information associated with orthologous genes from model organisms. The plugin annotates human variants and reports orthologous information from rat and mouse. The plugin is only available for GRCh38. The PhenotypeOrthologous file can be downloaded from https://ftp.ensembl.org/pub/current_variation/PhenotypeOrthologous The plugin can be run: `./vep -i variations.vcf --plugin PhenotypeOrthologous,file=PhenotypesOrthologous_homo_sapiens_112_GRCh38.gff3.gz` The file option is mandatory to run this plugin To return only results for rat : `./vep -i variations.vcf --plugin PhenotypeOrthologous,file=PhenotypesOrthologous_homo_sapiens_112_GRCh38.gff3.gz,model=rat` To return only results for mouse: `./vep -i variations.vcf --plugin PhenotypeOrthologous,file=PhenotypesOrthologous_homo_sapiens_112_GRCh38.gff3.gz,model=mouse` The tabix utility must be installed in your path to use this plugin. Check https://github.com/samtools/htslib.git for instructions. **Usage examples:** `mv PhenotypeOrthologous.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin PhenotypeOrthologous,file=PhenotypesOrthologous_homo_sapiens_112_GRCh38.gff3.gz`	Phenotype data and citations	-	Ensembl
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:	Phenotype data and citations	-	Ensembl

Plugin	Description	Category	External libraries	Developer

Argument	Description
`dir`	Path to directory where to look for phenotypes annotation. If the required file does not exist, the file is downloaded and saved in the provided directory (download requires using database or cache mode).
`file`	File path to phenotypes annotation. If the file does not exist, the file is downloaded and saved with this name (download requires using database or cache mode).

exclude_sources: &-separated list of phenotype sources to exclude. By default, HGMD-PUBLIC and COSMIC annotations are excluded. See http://www.ensembl.org/info/genome/variation/phenotype/sources_phenotype_documentation.html

include_sources: &-separated list of phenotype sources to include. If defined, exclude_sources is ignored.

exclude_types : &-separated list of feature types to exclude: Gene, Variation, QTL, StructuralVariation, SupportingStructuralVariation, RegulatoryFeature. By default, StructuralVariation and SupportingStructuralVariation annotations are always excluded (due to size issues) and Variation is excluded when annotating structural variants; to get these annotations in all cases, use include_types=StructuralVariation&SupportingStructuralVariation&Variation

include_types : &-separated list of feature types to include. If defined, exclude_types is ignored.

expand_right : Cache size in bp. By default, annotations 100000bp (100kb) downstream of the initial lookup are cached.

phenotype_feature : Boolean to report the gene/variation associated with the phenotype (such as overlapping gene or structural

```
                    variation) and annotation
source (default: 0)
```

cols : &-separated list of column and/or attribute names to output from the gff file. The output fields will be ordered in the same way given in cols argument. (default: `phenotype` or `source,phenotype,id` if you set phenotype_feature=1)

id_match : Return results only if the identifiers matches with the

```
                  variant or the gene depending
on the type (default: 0)
```

Example:

```
--plugin
Phenotypes,file=${HOME}/phenotypes.gff.gz,include_types=Gene
--plugin
Phenotypes,dir=${HOME},include_types=Gene
```

Usage examples:

```
mv Phenotypes.pm ~/.vep/Plugins
```

Plugin	Description	Category	External libraries	Developer
	``# Automatically download phenotype annotation files if needed and annotate # variants with phenotypes ./vep -i variations.vcf --plugin Phenotypes # Fetch only gene-associated phenotypes ./vep -i variations.vcf --plugin Phenotypes,include_types=Gene # Set directory with phenotypes annotations (phenotype annotation file is # automatically downloaded if not available in this directory) ./vep -i variations.vcf --plugin Phenotypes,dir=${HOME},include_types=Gene # Specify a file with phenotypes annotation (file is automatically # downloaded and saved with this name if it does not exist) ./vep -i variations.vcf --plugin Phenotypes,file=${HOME}/phenotypes.gff.gz,include_types=Gene```			
pLI	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). 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 pLI_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 https://ftp.broadinstitute.org/pub/ExAC_release/release0.3/functional_gene_constraint/fordist_cleaned_exac_r03_march16_z_pli_rec_null_data.txt From this file, extract gene or transcipt pLI scores: To extract gene scores : ```awk '{print $2, $20 }' fordist_cleaned_exac_r03_march16_z_pli_rec_null_data.txt > plI_gene.txt``` NB: The gene scores file can also be found in the VEP_plugins directory. To extract transcript scores: ```awk '{print $1, $20 }' fordist_cleaned_exac_r03_march16_z_pli_rec_null_data.txt > plI_transcript.txt``` NB: Using this file, No transcript score will be returned. To use another values file, add it as a parameter i.e.	Gene tolerance to change	• List::MoreUtils qw/zip/ • DBI	Ensembl

Plugin	Description	Category	External libraries	Developer
	``./vep -i variants.vcf --plugin pLI,values_file.txt ./vep -i variants.vcf --plugin pLI,values_file.txt,transcript # to check for the transcript score.``` gnomAD v4 release expanded the scale of pLI score calculation. The file can be downloaded from - https://gnomad.broadinstitute.org/downloads#v4-constraint (Constraint metrics TSV) To use the data you can follow the same procedure as above but needs to change the column number to accordingly. **Usage examples:** ```mv pLI.pm ~/.vep/Plugins mv pLI_values.txt ~/.vep/Plugins ./vep -i variants.vcf --plugin pLI```			
PolyPhen_SIFT	A VEP plugin that retrieves PolyPhen and SIFT predictions from a locally constructed SQLite database. It can be used when your main source of VEP transcript annotation (e.g. a GFF file or GFF-based cache) does not contain these predictions. You must create a SQLite database of the predictions or point to the SQLite database file already created. Compatible SQLite databases based on pangenome data are available at http://ftp.ensembl.org/pub/current_variation/pangenomes You may point to the file by adding parameter `db=[file]`. If the file is not in `HOME/.vep`, you can also use parameter `dir=[dir]` to indicate its path. ```--plugin PolyPhen_SIFT,db=[file] --plugin PolyPhen_SIFT,db=[file],dir=[dir]``` To create a SQLite database using PolyPhen/SIFT data from the Ensembl database, you must have an active database connection (i.e. not using `--offline`) and add parameter `create_db=1`. This will create a SQLite file named `[species].PolyPhen_SIFT.db`, placed in the directory specified by the `dir` parameter: ```--plugin PolyPhen_SIFT,create_db=1 --plugin PolyPhen_SIFT,create_db=1,dir=/some/specific/directory``` *** NB: this will take some hours! *** When creating a PolyPhen_SIFT by simply using `create_db=1`, you do not need to specify any parameters to load the appropriate file based on the species: ```--plugin PolyPhen_SIFT``` **Usage examples:** ```mv PolyPhen_SIFT.pm ~/.vep/Plugins # Read default PolyPhen/SIFT SQLite file in $HOME/.vep ./vep -i variations.vcf -cache --plugin PolyPhen_SIFT```	Pathogenicity predictions	• Digest::MD5 qw(md5_hex) • DBI	Ensembl

Plugin	Description	Category	External libraries	Developer
	<pre># Read database with custom name and/or located in a custom directory ./vep -i variations.vcf -cache --plugin PolyPhen_SIFT,db=custom.db ./vep -i variations.vcf -cache --plugin PolyPhen_SIFT,dir=/some/custom/dir ./vep -i variations.vcf -cache --plugin PolyPhen_SIFT,db=custom.db,dir=/some/custom/dir # Create PolyPhen/SIFT SQLite file based on Ensembl database ./vep -i variations.vcf -cache --plugin PolyPhen_SIFT,create_db=1</pre>			
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/. If you use this data, please cite the following publication Niroula, A., Vihinen, M. Harmful somatic amino acid substitutions affect key pathways in cancers. BMC Med Genomics 8, 53 (2015). https://doi.org/10.1186/s12920-015-0125-x There are two ways to run the plugin: 1. To compute the predictions from the PON-P2 API, use python script `ponp2.py` (*) and select the reference genome (acceptable values are: hg37 and hg38): <pre>--plugin PON_P2,pyscript=/path/to/python/script/ponp2.py,hg=hg37</pre> (*) To run this mode, 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. 2. To fetch the predictions from a file containing pre-calculated predictions for somatic variations please use the following key=value option (only available for GRCh37): Argument: `file` — Description: COSMIC text file with pre-calculated predictions downloaded from http://structure.bmc.lu.se/PON-P2/cancer30.html/ The following steps are necessary before using the file: <pre>(head -n 1 COSMIC.txt && tail -n +2 COSMIC.txt | sort -t $'\t' -k1,1 -k2,2n) > cosmic_sorted.txt sed -i 's/Chromosome/#Chromosome/' cosmic_sorted.txt bgzip cosmic_sorted.txt tabix -s 1 -b 2 -e 2 cosmic_sorted.txt.gz</pre>	Pathogenicity predictions	-	• Abhishek Niroula • Mauno Vihinen

Plugin	Description	Category	External libraries	Developer
	`--plugin PON_P2,file=path/to/cosmic_sorted.txt.gz` **Usage examples:** `mv PON_P2.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin PON_P2,pyscript=/path/to/python/script/ponp2.py,hg=hg37`			
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.	Phenotype data and citations	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`--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. **Usage examples:** `mv PostGAP.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin PostGAP,/path/to/PostGap.gz,col1,col2`			
PrimateAI	The PrimateAI VEP plugin is designed to retrieve clinical impact scores of variants, as described in https://www.nature.com/articles/s41588-018-0167-z. Please consider citing the paper if using this plugin. In brief, common missense mutations in non-human primate species are usually benign in humans. Thousands of common variants from six non-human primate species were used to train a deep neural network to identify pathogenic mutations in humans with a rare disease. This plugin uses files generated by the PrimateAI software, which is available from https://github.com/Illumina/PrimateAI. The files containing predicted pathogenicity scores can be downloaded from https://basespace.illumina.com/s/yYGFdGih1rXL (a free BaseSpace account may be required): PrimateAI_scores_v0.2.tsv.gz (for GRCh37/hg19) PrimateAI_scores_v0.2_hg38.tsv.gz (for GRCh38/hg38) Before running the plugin for the first time, the following steps must be taken to format the downloaded files: 1. Unzip the score files 2. Add '#' in front of the column description line 3. Remove any empty lines. 4. Sort the file by chromosome and position 5. Compress the file in .bgz format 6. Create tabix index (requires tabix to be installed). Command line examples for formatting input files: `gunzip -cf PrimateAI_scores_v0.2.tsv.gz | sed '12s/.*/#&/' | sed '/^$/d' | awk 'NR<12{print $0;next}{print $0 | "sort -k1,1 -k 2,2n -V"}' | bgzip > PrimateAI_scores_v0.2_GRCh37_sorted.tsv.bgz`	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	`tabix -s 1 -b 2 -e 2 PrimateAI_scores_v0.2_GRCh37_sorted.tsv.bgz` `gunzip -cf PrimateAI_scores_v0.2_hg38.tsv.gz | sed '12s/.*/#&/' | sed '/^$/d' | awk 'NR<12{print $0;next}{print $0 | "sort -k1,1 -k2,2n -V"}' | bgzip > PrimateAI_scores_v0.2_GRCh38_sorted.tsv.bgz` `tabix -s 1 -b 2 -e 2 PrimateAI_scores_v0.2_GRCh38_sorted.tsv.bgz` **Usage examples:** `mv PrimateAI.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin PrimateAI,PrimateAI_scores_v0.2_GRCh37_sorted.tsv.bgz` `./vep -i variations.vcf --plugin PrimateAI,PrimateAI_scores_v0.2_GRCh38_sorted.tsv.bgz`			
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. **Usage examples:** `mv ProteinSeqs.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin ProteinSeqs,reference.fa,mutated.fa` `./vep -i variations.vcf --plugin ProteinSeqs,reference=reference.fa,mutated=mutated.fa`	Sequence	-	Ensembl
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: `wget https://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/GRCh38/MISC/annotated_clone_assembly_problems_GCF_000001405.38.gff3` `wget https://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/`	Sequence	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	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: wget https://ftp.ncbi.nlm.nih.gov/pub/grc/human/GRC/GRCh37/MISC/annotated_clone_assembly_problems_GCF_000001405.25.gff3 wget https://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. **Usage examples:** mv ReferenceQuality.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin ReferenceQuality,/path/to/data.gff3.gz			
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 Running options: If available, the plugin will match the scores by transcript id (default). Using the flag 1 the plugin will not try to match by transcript id. REVEL scores can be downloaded from: https://sites.google.com/site/revelgenomics/downloads	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer
	The plugin supports several REVEL file versions: • REVEL file version Dec 2017, which has 7 columns and only GRCh37 coordinates • REVEL file version Feb 2020, which has 8 columns with GRCh37 and GRCh38 coordinates • REVEL file version May 2021, which has 9 columns with GRCh37 and GRCh38 coordinates and a new column with transcript ids These files can be tabix-processed by: `unzip revel-v1.3_all_chromosomes.zip` `cat revel_with_transcript_ids | tr "," "\t" > tabbed_revel.tsv` `sed '1s/.*/#&/' tabbed_revel.tsv > new_tabbed_revel.tsv` `bgzip new_tabbed_revel.tsv` for GRCh37: `tabix -f -s 1 -b 2 -e 2 new_tabbed_revel.tsv.gz` for GRCh38: `zcat new_tabbed_revel.tsv.gz | head -n1 > h` `zgrep -h -v ^#chr new_tabbed_revel.tsv.gz | awk '$3 != "." ' | sort -k1,1 -k3,3n - | cat h - | bgzip -c > new_tabbed_revel_grch38.tsv.gz` `tabix -f -s 1 -b 3 -e 3 new_tabbed_revel_grch38.tsv.gz` The plugin can then be run as default: `./vep -i variations.vf --assembly GRCh38 --plugin REVEL,file=/path/to/revel/data.tsv.gz` or with the option to not match by transcript id: `./vep -i variations.vf --assembly GRCh38 --plugin REVEL,file=/path/to/revel/data.tsv.gz,no_match=1` Requirements: The tabix utility must be installed in your path to use this plugin. The --assembly flag is required to use this plugin. **Usage examples:** `mv REVEL.pm ~/.vep/Plugins` `./vep -i variations.vf --assembly GRCh37 --plugin REVEL,file=/path/to/revel/data.tsv.gz` `./vep -i variations.vf --assembly GRCh38 --plugin REVEL,file=/path/to/revel/data.tsv.gz`			
RiboseqORFs	This is a VEP plugin that uses a standardized catalog of human Ribo-seq ORFs to re-calculate consequences for variants located in these translated regions.	Transcript annotation	-	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>This plugin reports new consequences based on the evidence from the Ribo-seq ORF annotation and supporting publications. The human Ribo-seq ORF data can be downloaded from: https://ftp.ebi.ac.uk/pub/databases/gencode/riboseq_orfs/data

After downloading the annotation, please bgzip and tabix it:

<pre>bgzip Ribo-seq_ORFs.bed
tabix Ribo-seq_ORFs.bed.gz</pre>
For optimal performance when running this plugin in VEP, please use a FASTA file (<code>--fasta</code>). A FASTA file is always required in offline mode.

Please cite the publication for the Ribo-seq ORF annotation alongside the VEP if you use this resource: https://doi.org/10.1038/s41587-022-01369-0

The tabix utility must be installed in your path to use this plugin.

Usage examples:

<pre>./vep -i variations.vcf --plugin RiboseqORFs,file=/path/to/Ribo-seq_ORFs.bed.gz</pre></td><td></td><td></td><td></td></tr>
<tr><td>SameCodon</td><td>A VEP plugin that reports existing variants that fall in the same codon. This plugin requires a database connection, can not be run in offline mode

Usage examples:

<pre>mv SameCodon.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin SameCodon</pre></td><td>Variant data</td><td>-</td><td>Ensembl</td></tr>
<tr><td>satMutMPRA</td><td>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:

<table><tr><th>Argument</th><th>Description</th></tr><tr><td><code>file</code></td><td>required - a tabix indexed file of the satMutMPRA data corresponding to desired assembly.</td></tr></table></td><td>Phenotype data and citations</td><td>-</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer

Argument	Description
`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
```

Plugin	Description	Category	External libraries	Developer
	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. **Usage examples:** `mv satMutMPRA.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin satMutMPRA,file=/path/to/satMutMPRA_data.gz,cols=col1:col2`			
SingleLetterAA	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that returns a HGVSp string with single amino acid letter codes **Usage examples:** `mv SingleLetterAA.pm ~/.vep/Plugins` `./vep -i variations.vcf --plugin SingleLetterAA`	HGVS	-	Ensembl
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: 1. By default, this plugin appends all scores from SpliceAI files.	Splicing predictions	List::Util qw(max)	Ensembl

Plugin	Description	Category	External libraries	Developer

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 the output contains one header `SpliceAI_pred` with all the delta scores and positions. The format is: `SYMBOL|DS_AG|DS_AL|DS_DG|DS_DL|DP_AG|DP_AL|DP_DG|DP_DL`
- For JSON the output is a hash with the following format: "spliceai": {"DP_DL":0,"DS_AL":0,"DP_AG":0,"DS_DL":0,"SYMBOL":"X","DS_AG":0,"DP_AL":0,"DP_DG":0,"DS_DG":0}
- 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.

Gene matching: SpliceAI can contain scores for multiple genes that overlap a variant, and VEP can also predict consequences on multiple genes for a given variant. The plugin only returns SpliceAI scores for the gene symbols that match (if any).

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.h
g38.vcf.gz,indel=/path/to/spliceai_scores.raw.i
ndel.hg38.vcf.gz
./vep -i variations.vcf --plugin
SpliceAI,snv=/path/to/spliceai_scores.raw.snv.h
g38.vcf.gz,indel=/path/to/spliceai_scores.raw.i
ndel.hg38.vcf.gz,cutoff=0.5
```

Usage examples:

```
mv SpliceAI.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
```

Plugin	Description	Category	External libraries	Developer
	SpliceAI,snv=/path/to/spliceai_snv_.vcf.gz, indel=/path/to/spliceai_indel_.vcf.gz			
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 vvvvvvvvvvvvvvv ...IIIIIIIIIIIIIIIIIIIEEEEE... **Usage examples:** mv SpliceRegion.pm ~/.vep/Plugins ./vep -i variations.vcf --plugin SpliceRegion To only show the additional consequence extended_intronic_splice_region_variant, use: ./vep -i variations.vcf --plugin SpliceRegion,Extended	Splicing predictions	-	Ensembl
SpliceVault	A VEP plugin that retrieves SpliceVault data to predict exon-skipping events and activated cryptic splice sites based on the most common mis-splicing events around a splice site. This plugin returns the most common variant-associated mis-splicing events based on SpliceVault data. Each event includes the following information: - Type: exon skipping (ES), cryptic donor (CD) or cryptic acceptor (CA) - Transcript impact: - For ES, describes skipped exons, e.g. ES:2 represents exon 2 skipping and ES:2-3 represents skipping of exon 2 and 3 - For CD/CA, describes the distance from the annotated splice-site to the cryptic splice-site with reference to the transcript (distances to negative strand transcripts are reported according to the 5' to 3' distance) - Percent of supporting samples: percent of samples supporting the event over total samples where splicing occurs in that site (note this may be above 100% if the event is seen in more samples than annotated splicing) - Frameshift: inframe or out-of-frame event	Splicing predictions	-	Ensembl

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>
The plugin also returns information specific to each splice site:

Site position/type: genomic location and type (donor/acceptor) of the splice-site predicted to be lost by SpliceAI. Cryptic positions are relative to this genomic coordinate.
Out of frame events: fraction of the top events that cause a frameshift. As per https://pubmed.ncbi.nlm.nih.gov/36747048, sites with 3/4 or more in-frame events are likely to be splice-rescued and not loss-of-function (LoF).
Site sample count and max depth: sample count for this splice site and max number of reads in any single sample representing annotated splicing in Genotype-Tissue Expression (GTEx). This information allows to filter events based on a minimum number of samples or minimum depth in GTEx.
SpliceAI delta score (provided by SpliceVault)

Please cite the SpliceVault publication alongside the VEP if you use this resource: https://pubmed.ncbi.nlm.nih.gov/36747048

The tabix utility must be installed in your path to use this plugin. The SpliceVault TSV and respective index (TBI) for GRCh38 can be downloaded from:

https://ftp.ensembl.org/pub/current_variation/SpliceVault/SpliceVault_data_GRCh38.tsv.gz
https://ftp.ensembl.org/pub/current_variation/SpliceVault/SpliceVault_data_GRCh38.tsv.gz.tbi

To filter results, please use filter_vep with the output file or standard output. Documentation on filter_vep is available at: https://www.ensembl.org/info/docs/tools/vep/script/vep_filter.html

Usage examples:
<pre>
mv SpliceVault.pm ~/.vep/Plugins

./vep -i variations.vcf --plugin SpliceVault,file=/path/to/SpliceVault_data_GRCh38.tsv.gz

Stringently select predicted loss-of-function (pLoF) splicing variants
./filter_vep -i variant_effect_output.txt --filter "SPLICEVAULT_OUT_OF_FRAME_EVENTS >= 3"
</pre>
</td><td></td><td></td><td></td></tr>
<tr><td>StructuralVariantOverlap</td><td>
A VEP plugin that retrieves information from overlapping structural variants.

Parameters can be set using a key=value system:
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>file</td><td>required - a VCF file of reference data.</td></tr>
<tr><td>percentage</td><td>percentage overlap between SVs (default: 80)</td></tr>
</table>
</td><td>Structural variant data</td><td>-</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer

Argument	Description
`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: https://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/gcp-public-data--gnomad/papers/2019-sv/gnomad_v2.1_sv.sites.vcf.gz

Example:

```
./vep -i structvariants.vcf --plugin
StructuralVariantOverlap,file=gnomad_v2_sv.site
s.vcf.gz
```

Usage examples:

```
mv StructuralVariantOverlap.pm ~/.vep/Plugins
./vep -i structvariants.vcf --plugin
StructuralVariantOverlap,file=gnomad_v2_sv.site
s.vcf.gz
```

Plugin	Description	Category	External libraries	Developer
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. The plugin accepts the following key=value parameters:	Variant data	• Data::Dumper • Storable qw(dclone)	Joseph A. Prinz

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>name</code></td><td>short name added used as a prefix (required)</td></tr>
<tr><td><code>file</code></td><td>path to tabix-index vcf file (required)</td></tr>
<tr><td><code>filter</code></td><td>only consider variants marked as <code>PASS</code>, 1 or 0 (default, 1)</td></tr>
<tr><td><code>fields</code></td><td>info fields to be returned (default, not used)
'%' can delimit multiple fields
'*' can be used as a wildcard</td></tr>
</table>
Returns:

<code><name>_POS</code>: POS field from VCF
<code><name>_REF</code>: REF field from VCF (minimised)
<code><name>_ALT</code>: ALT field from VCF (minimised)
<code><name>_alt_index</code>: Index of matching variant (zero-based)
<code><name>_<field></code>: List of requested info values

<h3>Usage examples:</h3>
<pre><code>./vep -i variations.vcf --plugin
SubsetVCF,file=filepath.vcf.gz,name=myvfc,field
s=AC*%AN*</code></pre>
</td><td></td><td></td><td></td></tr>
<tr><td>TranscriptAnnotator</td><td>A VEP plugin that annotates variant-transcript pairs based on a given file:
<pre><code>--plugin
TranscriptAnnotator,file=${HOME}/file.tsv.gz</code></pre>
Example of a valid tab-separated annotation file:
<pre><code>#Chrom Pos   Ref Alt Transcript
SIFT_score SIFT_pred   Comment
11   436154   A   G   NM_001347882.2
0.03   Deleterious Bad
11   1887471   C   T   ENST00000421485
0.86   Tolerated   Good</code></pre>
Please bgzip and tabix the file with commands such as:
<pre><code>bgzip file.txt
tabix -b2 -e2 file.txt.gz</code></pre>
Options are passed to the plugin as key=value pairs:
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>file</code></td><td>(mandatory) Tabix-indexed file to parse. Must contain variant location (chromosome, position, reference allele, alternative allele) and transcript ID as the first 5 columns. Accepted transcript IDs include those from Ensembl and RefSeq.</td></tr>
</table>
</td><td>Transcript annotation</td><td>File::Basename</td><td>Ensembl</td></tr>
</table>

<table>
<tr><th>Plugin</th><th>Description</th><th>Category</th><th>External libraries</th><th>Developer</th></tr>
<tr><td></td><td>
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>cols</code></td><td>Colon-delimited list with names of the columns to append. Column names are based on the last header line. By default, all columns (except the first 5) are appended.</td></tr>
<tr><td><code>prefix</code></td><td>String to prefix the name of appended columns (default: basename of the filename without extensions). Set to 0 to avoid any prefix.</td></tr>
<tr><td><code>trim</code></td><td>Trim whitespaces from both ends of each column (default: 1).</td></tr>
</table>
The tabix and bgzip utilities must be installed in your path to read the tabix-indexed annotation file: check https://github.com/samtools/htslib.git for installation instructions.
<h3>Usage examples:</h3>
<pre><code>mv TranscriptAnnotator.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin
TranscriptAnnotator,file=/path/to/file.txt.gz</code></pre>
</td><td></td><td></td><td></td></tr>
<tr><td>TSSDistance</td><td>A VEP plugin that calculates the distance from the transcription start site for upstream variants.
<h3>Usage examples:</h3>
<pre><code>mv TSSDistance.pm ~/.vep/Plugins
./vep -i variations.vcf --plugin TSSDistance</code></pre>
</td><td>Nearby features</td><td>-</td><td>Ensembl</td></tr>
<tr><td>UTRAnnotator</td><td>A VEP plugin that annotates the effect of 5' UTR variant especially for variant creating/disrupting upstream ORFs. Available for both GRCh37 and GRCh38.
Options are passed to the plugin as key=value pairs:
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>file</code></td><td>(Required) Path to UTRAnnotator data file:
Download <code>uORF_5UTR_GRCh37_PUBLIC.txt</code> or <code>uORF_5UTR_GRCh38_PUBLIC.txt</code> from https://github.com/Ensembl/UTRannotator
Download from http://sorfs.org</td></tr>
<tr><td><code>max_overlap</code></td><td>(Optional) Maximum percentage of overlap between variant and UTR for UTR annotation (default: 100)</td></tr>
</table>
Citation
About the role of 5'UTR variants in human genetic disease:
Whiffin, N., Karczewski, K.J., Zhang, X. et al. Characterising the loss-of-function impact of 5’ untranslated region variants in 15,708 individuals. Nat Commun 11, 2523 (2020). https://doi.org/10.1038/s41467-019-10717-9
About UTRAnnotator:
</td><td>Transcript annotation</td><td>List::Util qw(min max)
Scalar::Util qw(looks_like_number)</td><td>Ensembl</td></tr>
</table>

Plugin	Description	Category	External libraries	Developer
	The original UTRAnnotator plugin is written by Xiaolei Zhang et al. Later adopted by Ensembl VEP plugins with some changes. You can find the original plugin here - https://github.com/ImperialCardioGenetics/UTRannotator Please cite the UTRannotator publication alongside the Ensembl VEP if you use this resource - Annotating high-impact 5'untranslated region variants with the UTRannotator Zhang, X., Wakeling, M.N., Ware, J.S, Whiffin, N. Bioinformatics; doi: https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btaa783/5905476 **Usage examples:** `mv UTRAnnotator.pm ~/.vep/Plugins` `vep -i variations.vcf --plugin UTRAnnotator,file=/path/to/uORF_starts_ends_GRCh38_PUBLIC.txt` `# skip annotation for variants with a 80% or higher overlap of the UTR` `vep -i variations.vcf --plugin UTRAnnotator,file=/path/to/uORF_starts_ends_GRCh38_PUBLIC.txt,max_overlap=80`			
VARITY	This is a plugin for the Ensembl Variant Effect Predictor (VEP) that adds the pre-computed VARITY scores to predict pathogenicity of rare missense variants to VEP output. Please cite the VARITY publication alongside the VEP if you use this resource: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8715197/ Running options : VARITY scores can be downloaded using `wget http://varity.varianteffect.org/downloads/varity_all_predictions.tar.gz` The files can be tabix processed by : `tar -xzvf varity_all_predictions.tar.gz` `cat varity_all_predictions.txt | (head -n 1 && tail -n +2 | sort -t$'\t' -k 1,1 -k 2,2n) > varity_all_predictions_sorted.tsv` `sed '1s/.*/#&/' varity_all_predictions_sorted.tsv > varity_all_predictions.tsv  # to add a # in the first line of the file` `bgzip varity_all_predictions.tsv` `tabix -f -s 1 -b 2 -e 2 varity_all_predictions.tsv.gz` Requirements: The tabix utility must be installed in your path to use this plugin. The --assembly flag is required to use this plugin. **Usage examples:** `mv VARITY.pm ~/.vep/Plugins` `./vep -i variations.vcf --assembly GRCh37 --plugin VARITY,file=/path/to/varity_all_predictions.txt`	Pathogenicity predictions	-	Ensembl

Plugin	Description	Category	External libraries	Developer

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.

There are further published plugins available outside the VEP repository including:

- LOFTEE a Loss-Of-Function Transcript Effect Estimator (Konrad Karczewski et al,2020)

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'];
}
```

Variant Effect Predictor — Examples and use cases

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
file=ftp://ftp.myhost.org/data/phenotypes.bed.gz,short_name=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
```

- Report uploaded allele before minimisation. For more information, see here.

```
./vep --cache -i variants.vcf --uploaded_allele
```

---

gnomAD

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

For the latest gnomAD data, please visit gnomAD downloads.

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
   file=gnomad.genomes.r2.0.1.sites.GRCh38.noVEP.vcf.gz,short_name=gnomADg,format=vcf,type=exact,c
   oords=0,fields=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 file=All_hg19_RS.bw,short_name=GERP,format=bigwig
```

Example conservation score files:

Human (GRCh38)

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

Human (GRCh37)

- GERP
- phastCons 46-way
- phastCons 100-way
- phyloP 46-way
- phyloP 100-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, LRT, MutationTaster, FATHMM) across every possible missense substitution in the human proteome.

Plugins in VEP sometimes require data processed in specific ways as arguments. Any requirements and usage instructions for each plugin can be found in the plugin documentation.

In the case of the dbNSFP.pm plugin, the data needs to be downloaded and then processed into a format that the plugin can use. Note that there are two distinct branches of the files provided for academic and commercial usage; please use the appropriate files for your use case.

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.

```
version=4.5c
unzip dbNSFP${version}.zip
zcat dbNSFP${version}_variant.chr1.gz | head -n1 > h

# GRCh38/hg38 data
zgrep -h -v "^#chr" dbNSFP${version}_variant.chr* | sort -k1,1 -k2,2n - | cat h - | bgzip -c >
dbNSFP${version}_grch38.gz
tabix -s 1 -b 2 -e 2 dbNSFP${version}_grch38.gz

# GRCh37/hg19 data
zgrep -h -v "^#chr" dbNSFP${version}_variant.chr* | awk '$8 != "." ' | sort -k8,8 -k9,9n - | cat h
- | bgzip -c > dbNSFP${version}_grch37.gz
tabix -s 8 -b 9 -e 9 dbNSFP${version}_grch37.gz
```

Then simply download the dbNSFP.pm 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,dbNSFP4.5c_grch38.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, 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,dbNSFP4.5c_grch38.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 (SV) with their predicted effect on other genomic features. For more information on SV input format, see here.

Prediction process

- If the INFO keys `END` or `SVLEN` are present, the proportion of any overlapping feature covered by the variant is calculated
- The alternative allele (or `SVTYPE` in older VCF files) defines the type of structural variant; some types of structural variants are tested for specific consequences:

Structural variant type	Abbreviation	Specific consequences
Insertion	INS	Feature elongation
Deletion	DEL	Feature truncation
Duplication	DUP	Feature amplification/elongation
Inversion	INV	*Not tested for any specific consequence*
Copy number variation	CNV	Feature amplification/elongation (if copy number is 2) or truncation (if copy number is 0)
Breakpoint variant	BND	Feature truncation

Insertions and deletions

- Supports mobile element insertions/deletions, including ALU, HERV, LINE1 and SVA elements
  - Currently, mobile element variants are treated as any insertion/deletion

Breakpoint variants

- Supports chromosome synonyms in breakends (such as `chr4` and `NC_000004.12`)
- Processes single breakends and multiple, comma-separated alternative breakends
- Consequences are reported for each breakend; for instance, for a VCF input like `1 7936271 . N N[12:58877476[,N[X:10932343[`, it will report the consequences for each of the 3 breakends:
  - `N[12:58877476[`: consequences for the first alternative breakend near chr12:58877476
  - `N[X:10932343[`: consequences for the second alternative breakend near chrX:10932343
  - `N.`: consequences for the reference breakend near chr1:7936271 (represented as detailed in the VCF 4.4 specification, section 5.4.9: Single breakends)
- In case of specific breakends not overlaping any reported Ensembl features (such as transcripts and regulatory regions), that specific breakend will **NOT** be presented in VEP output.

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.

Plugin support

- CADD plugin
- Conservation plugin
- NearestGene plugin
- Phenotypes plugin
- StructuralVariantOverlap plugin: please note that all features of this plugin have been ported to --custom annotation, with additional improvements
- TSSDistance plugin

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.

- This limit is not associated with breakpoint variants: each breakend in a breakpoint variant is analysed by VEP as a single base (the alternative sequence is currently ignored).

- 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.

Pangenome assemblies

VEP is able to analyse variants in **any species or assembly** (even if not part of Ensembl data) by providing your own FASTA file and GFF/GTF annotation:

```
./vep -i variants.txt -o variants_output.txt --gff data.gff.gz --fasta genome.fa.gz
```

We also provide data for other assemblies besides those supported in the current Ensembl and Ensembl Genomes sites.

HPRC assemblies

The Human Pangenome Reference Consortium (HPRC) aims to sequence 350 individuals of diverse ancestries, producing a pangenome of 700 haplotypes by the end of 2024. The first publication (*A draft human pangenome reference*) describes 47 phased, diploid assemblies from a cohort of genetically diverse individuals.

The VEP command-line tool (CLI) can annotate and filter variants called against the latest human assemblies, including the telomere-to-telomere assembly of the CHM13 cell line (T2T-CHM13). We have annotated genes on these human assemblies, based on Ensembl/GENCODE 38 genes and transcripts, via a new mapping pipeline as detailed in the Methods section of *A draft human pangenome reference*. The links to download and visualise the human annotations for HPRC assemblies are summarised in the Ensembl HPRC data page.

Running VEP with HPRC assemblies

Currently, VEP can only be run with HPRC assemblies in offline mode, one assembly at a time. There are two ways to use VEP with HPRC assemblies:

- Using **VEP cache** with (recommended) **FASTA sequence** (the most efficient way)
- Using **GTF annotation** with (mandatory) **FASTA sequence**

In the examples below, we demonstrate annotating variants on **T2T-CHM13v2.0** (GCA_009914755.4 assembly). To create a sample VCF to use in the examples below, you can take the first 100 lines from the ClinVar VCF file mapped to T2T-CHM13:

```
clinvar=ftp://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/variation/2022_10/vcf/2024_07/clinvar_2024062
4_GCA_009914755.4.vcf.gz
tabix -h $clinvar 1 | head -n 100 > test.vcf
```

VEP cache

VEP cache is a downloadable archive containing all transcript models for an assembly; it may also contain regulatory features and variant data.

Let's start by downloading and extracting the VEP cache to the default VEP directory (available for each annotation by clicking in **VEP cache** in the Ensembl HPRC data page). In the case of T2T-CHM13:

```
cd $HOME/.vep
curl -O https://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/variation/2022_10/indexed_vep_cache/Homo_sapi
ens-GCA_009914755.4-2022_10.tar.gz
tar xzf Homo_sapiens-GCA_009914755.4-2022_10.tar.gz
```

This will create the folder `homo_sapiens_gca009914755v4/107_T2T-CHM13v2.0` with the gene data required to run VEP. The name of this folder contains relevant information when running VEP:

- Species: `homo_sapiens_gca009914755v4`
- Cache version: `107`
- Assembly: `T2T-CHM13v2.0`

As well as molecular consequence predictions, many gene/transcript-based VEP options are supported for HPRC assemblies:

```
vep -i test.vcf --offline \
    --species homo_sapiens_gca009914755v4 \
    --cache_version 107 \
    --fasta Homo_sapiens-GCA_009914755.4-softmasked.fa.gz \
    --domains --symbol --canonical --protein --biotype --uniprot --variant_class
```

We don't have other annotations, such as RefSeq transcripts or variant information in the cache.

To run VEP with the downloaded cache in offline mode, please specify the species (which here includes assembly name) and cache version:

```
vep -i test.vcf --offline --species homo_sapiens_gca009914755v4 --cache_version 107
```

FASTA sequence

When using VEP cache, supplying the reference genomic sequence in a FASTA file is optional, but is required to enable the following options:

- Create HGVS notations (--hgvs and --hgvsg)
- Check the reference sequence given in input data (--check_ref)

Genomic FASTA files can be found in Ensembl HPRC data page **> FTP dumps > ensembl > genome**. FASTA files need to be either uncompressed or compressed with **bgzip** (recommended) to be compatible with VEP. For instance, to download a compressed FASTA file, uncompress it and then re-compress it with bgzip:

```
curl -O https://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/genome/Homo_sapiens-GCA_009914755.4-
softmasked.fa.gz
gzip -d Homo_sapiens-GCA_009914755.4-softmasked.fa.gz
bgzip Homo_sapiens-GCA_009914755.4-softmasked.fa.gz
```

Afterwards, you can run VEP using cache and the --fasta flag:

```
vep -i test.vcf --offline \
    --species homo_sapiens_gca009914755v4 \
    --cache_version 107 \
    --fasta Homo_sapiens-GCA_009914755.4-softmasked.fa.gz
```

More information on using FASTA files with VEP is available here.

GTF and GFF annotation

As an alternative to using cache files, VEP can utilise gene information in appropriately indexed GTF or GFF files. GTF and GFF files can be downloaded from the annotation column in the Ensembl HPRC data page. The data needs to be re-sorted in chromosomal order, compressed in **bgzip** and indexed with **tabix**. We present here the example for a GTF file:

```
curl -O https://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/geneset/2022_07/Homo_sapiens-GCA_009914755.4-
2022_07-genes.gtf.gz
gzip -d Homo_sapiens-GCA_009914755.4-2022_07-genes.gtf.gz
grep -v "#" Homo_sapiens-GCA_009914755.4-2022_07-genes.gtf |\
  sort -k1,1 -k4,4n -k5,5n -t$'\t' |\
  bgzip -c > Homo_sapiens-GCA_009914755.4-2022_07-genes.gtf.gz
tabix Homo_sapiens-GCA_009914755.4-2022_07-genes.gtf.gz
```

FASTA files are **always** required when running HPRC data with GTF annotation, as the transcript sequences are not available in the GFF files.

Afterwards, you can run VEP using the GTF and FASTA files:

```
vep -i test.vcf \
      --gtf Homo_sapiens-GCA_009914755.4-2022_07-genes.gtf.gz \
      --fasta Homo_sapiens-GCA_009914755.4-softmasked.fa.gz
```

Check here for more information on using VEP with GTF and GFF annotation.

Missense deleteriousness predictions

Although PolyPhen/SIFT scores are not directly available for alternative assemblies by using --polyphen and --sift, they can be retrieved via the PolyPhen_SIFT plugin.

Using our ProteinFunction pipeline, we ran **PolyPhen-2 2.2.3** and **SIFT 6.2.1** on the proteome sequences for GRCh38 and all HPRC assemblies (the protein FASTA files indicated in Ensembl HPRC data page) and stored their results in a single SQLite file: homo_sapiens_pangenome_PolyPhen_SIFT_20240502.db.

Pre-computed scores and predictions can be retrieved by downloading this file and running VEP with the **PolyPhen_SIFT plugin**:

```
curl -O
https://ftp.ensembl.org/pub/current_variation/pangenomes/Human/homo_sapiens_pangenome_PolyPhen_SIF
T_20240502.db
vep -i test.vcf --offline \
    --species homo_sapiens_gca009914755v4 \
    --cache_version 107 \
    --fasta Homo_sapiens-GCA_009914755.4-softmasked.fa.gz \
    --plugin PolyPhen_SIFT,db=human_pangenomes.PolyPhen_SIFT.db
```

Matched variant annotations (ClinVar, gnomAD and dbSNP)

We don't have variant data in the VEP caches for the pangenome assemblies, but it can be integrated using the --custom option with data files using the same assembly coordinates. We have lifted-over some key datasets, including ClinVar and gnomAD to the HPRC assemblies (downloadable from the VCF column in Ensembl HPRC data page).

```
# Download ClinVar data and respective index (TBI)
curl -O https://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/variation/2022_10/vcf/2024_07/clinvar_2024062
4_GCA_009914755.4.vcf.gz
curl -O https://ftp.ensembl.org/pub/rapid-
release/species/Homo_sapiens/GCA_009914755.4/ensembl/variation/2022_10/vcf/2024_07/clinvar_2024062
4_GCA_009914755.4.vcf.gz.tbi

# Run VEP with ClinVar data
vep -i test.vcf --offline \
    --species homo_sapiens_gca009914755v4 --cache_version 107 \
    --fasta Homo_sapiens-GCA_009914755.4-softmasked.fa.gz \
    --custom
file=clinvar_20240624_GCA_009914755.4.vcf.gz,short_name=ClinVar,format=vcf,type=exact,coords=0,fie
lds=CLNSIG%CLNREVSTAT%CLNDN
```

Additional annotations

Ensembl VEP plugins are a simple way to add new functionality to your analysis. Many require data that is only available for GRCh37 or GRCh38, but others, for example those based on gene attributes or on the fly analysis are compatible with the HGRC assemblies.

Here follows VEP plugins that are easily compatible with alternative human assemblies:

Plugin	Description	Plugin data	Usage example
Blosum62	Looks up the BLOSUM 62 substitution matrix score for the reference and alternative amino acids predicted for a missense mutation.		`--plugin Blosum62`
DosageSensitivity	Retrieves haploinsufficiency and triplosensitivity probability scores for affected genes (Collins *et al.*, 2022).	Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz	`--plugin DosageSensitivity,file=Collins_rCNV_2022.dosage_sensitivity_scores.tsv.gz`
Downstream	Predicts downstream effects of a frameshift variant on the protein sequence of a transcript.	Requires a FASTA file provided via the --fasta option	`--plugin Downstream`
Draw	Draws pictures of the transcript model showing the variant location.		`--plugin Draw`

Plugin	Description	Plugin data	Usage example
GeneSplicer	Runs GeneSplicer to get splice site predictions.	Binary and training data for GeneSplicer (plugin instructions)	`--plugin GeneSplicer,binary=genesplicer/bin/linux/genesplicer,training=genesplicer/human`
GO	Retrieves Gene Ontology (GO) terms associated with genes (for HGRC assemblies, specifically) using custom GFF annotation containing GO terms.	Ensembl HPRC data page > **FTP dumps > ensembl > variation > [date] > gff:** • `*_GO_plugin.gff.gz` • `*_GO_plugin.gff.gz.tbi`	`--plugin GO,file=homo_sapiens_gca009914755v4_110_VEP_GO_plugin.gff.gz`
HGVSIntronOffset	Returns HGVS intron start and end offsets. To be used with --hgvs option.		`--plugin HGVSIntronOffset`
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.		`--plugin LoFtool`
MaxEntScan	Runs MaxEntScan to get splice site predictions.	Extracted directory from fordownload.tar.gz	`--plugin MaxEntScan,/path/to/fordownload`
NearestExonJB	Finds the nearest exon junction boundary to a coding sequence variant.		`--plugin NearestExonJB`
NMD	Predicts if a variant allows the transcript to escape nonsense-mediated mRNA decay based on certain rules.		`--plugin NMD`
Phenotypes	Retrieves overlapping phenotype information.	Ensembl HPRC data page > **FTP dumps > ensembl > variation > [date] > gff:** • `*_phenotypes_plugin.gvf.gz` • `*_phenotypes_plugin.gvf.gz.tbi`	`--plugin Phenotypes,file=homo_sapiens_gca009914755v4_110_VEP_phenotypes_plugin.gvf.gz`
pLI	Adds the probability of a gene being loss-of-function intolerant (pLI).		`--plugin pLI`
PolyPhen_SIFT	Retrieves PolyPhen and SIFT predictions from a SQLite database.	homo_sapiens_pangenome_PolyPhen_SIFT_20240502.db	`--plugin PolyPhen_SIFT,db=homo_sapiens_pangenome_PolyPhen_SIFT_20240502.db`
ProteinSeqs	Writes two files with the reference and mutated protein sequences of any proteins found with non-synonymous mutations in the input file.		`--plugin ProteinSeqs`
SingleLetterAA	Returns HGVSp string with single amino acid letter codes.		`--plugin SingleLetterAA`
SpliceRegion	Provides more granular predictions of splicing effects.		`--plugin SpliceRegion`
SubsetVCF	Retrieves overlapping records from a given VCF file.	A VCF file	`--plugin SubsetVCF,file=file.vcf.gz,name=myvfc`

Plugin	Description	Plugin data	Usage example
TranscriptAnnotator	Annotates variant-transcript pairs based on a given file.	Tab-separated annotation file (plugin instructions)	`--plugin TranscriptAnnotator,file=annotation.txt.gz`
TSSDistance	Calculates the distance from the transcription start site for upstream variants.		`--plugin TSSDistance`

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 ℹ 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 (114) 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  Codons  Existing_variation  Extra
rs699  1:230710048  G  ENSG00000135744  ENST00000366667  Transcript  missense_variant  1018
803  268  M/T  aTg/aCg  -  IMPACT=MODERATE;STRAND=-1
rs699  1:230710048  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -
-  -  -  -  -  IMPACT=MODIFIER;DISTANCE=650;STRAND=-1
var_2  1:230710514  G  ENSG00000135744  ENST00000366667  Transcript  synonymous_variant  552
337  113  L  Ttg/Ctg  -  IMPACT=LOW;STRAND=-1
var_2  1:230710514  T  ENSG00000135744  ENST00000366667  Transcript  missense_variant  552
337  113  L/M  Ttg/Atg  -  IMPACT=MODERATE;STRAND=-1
var_2  1:230710514  G  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -
-  -  -  -  -  IMPACT=MODIFIER;DISTANCE=184;STRAND=-1
var_2  1:230710514  T  ENSG00000244137  ENST00000412344  Transcript  downstream_gene_variant  -
-  -  -  -  -  IMPACT=MODIFIER;DISTANCE=184;STRAND=-1
```

Options

- **--pick**

  VEP chooses one block of annotation per variant, using an ordered set of criteria. This order may be customised using --pick_order.

  1. MANE Select transcript status

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

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        843     776     259     M/T     aTg/aCg -
IMPACT=MODERATE;STRAND=-1
var_2   1:230710514     T       ENSG00000135744 ENST00000366667 Transcript
missense_variant        377     310     104     L/M     Ttg/Atg -
IMPACT=MODERATE;STRAND=-1
```

- **--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        843     776     259     M/T     aTg/aCg -
IMPACT=MODERATE;STRAND=-1
var_2   1:230710514     T       ENSG00000135744 ENST00000366667 Transcript
missense_variant        377     310     104     L/M     Ttg/Atg -
IMPACT=MODERATE;STRAND=-1
var_2   1:230710514     G       ENSG00000135744 ENST00000366667 Transcript
synonymous_variant      377     310     104     L       Ttg/Ctg -       IMPACT=LOW;STRAND=-1
```

- **--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        843     776     259     M/T     aTg/aCg -
IMPACT=MODERATE;STRAND=-1
rs699   1:230710048     G       ENSG00000244137 ENST00000412344 Transcript
downstream_gene_variant -       -       -       -       -       -
IMPACT=MODIFIER;DISTANCE=650;STRAND=-1
var_2   1:230710514     T       ENSG00000135744 ENST00000366667 Transcript
missense_variant        377     310     104     L/M     Ttg/Atg -
IMPACT=MODERATE;STRAND=-1
var_2   1:230710514     G       ENSG00000244137 ENST00000412344 Transcript
downstream_gene_variant -       -       -       -       -       -
IMPACT=MODIFIER;DISTANCE=184;STRAND=-1
```

- **--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       ENSG00000135744 ENST00000366667 Transcript
missense_variant        843     776     259     M/T     aTg/aCg -
IMPACT=MODERATE;STRAND=-1
rs699   1:230710048     G       ENSG00000244137 ENST00000412344 Transcript
downstream_gene_variant -       -       -       -       -       -
IMPACT=MODIFIER;DISTANCE=650;STRAND=-1
var_2   1:230710514     T       ENSG00000135744 ENST00000366667 Transcript
missense_variant        377     310     104     L/M     Ttg/Atg -
IMPACT=MODERATE;STRAND=-1
var_2   1:230710514     T       ENSG00000244137 ENST00000412344 Transcript
downstream_gene_variant -       -       -       -       -       -
IMPACT=MODIFIER;DISTANCE=184;STRAND=-1
var_2   1:230710514     G       ENSG00000135744 ENST00000366667 Transcript
synonymous_variant      377     310     104     L       Ttg/Ctg -       IMPACT=LOW;STRAND=-1
var_2   1:230710514     G       ENSG00000244137 ENST00000412344 Transcript
downstream_gene_variant -       -       -       -       -       -
IMPACT=MODIFIER;DISTANCE=184;STRAND=-1
```

- **--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.

Reference sequence used as part of VEP's HGVSc calculations is taken from a given FASTA file, rather than the variant reference. HGVSp is calculated using the given variant reference.

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. We do not support predicted RefSeq transcripts for GRCh37

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.

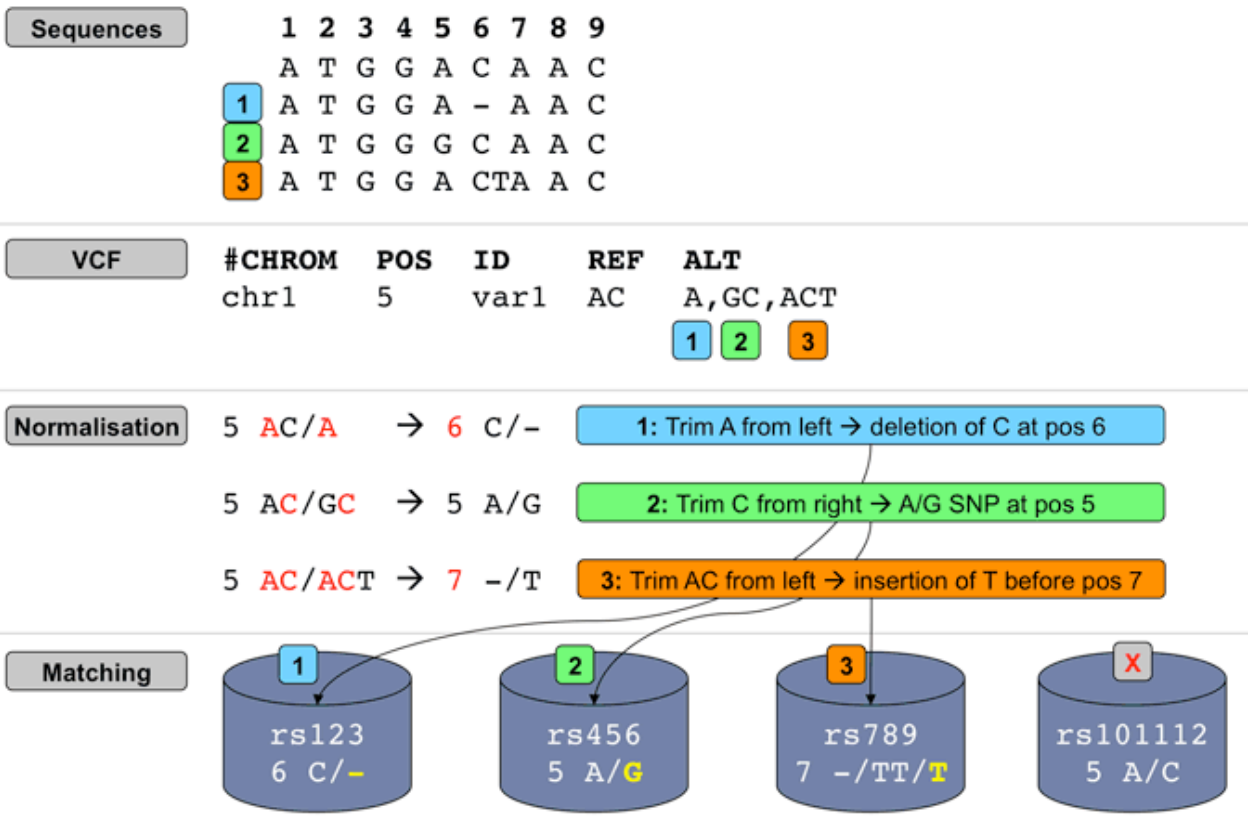


Illustration of VEP's allele matching algorithm resolving one VCF line with multiple ALTs to three different variant types and coordinates

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

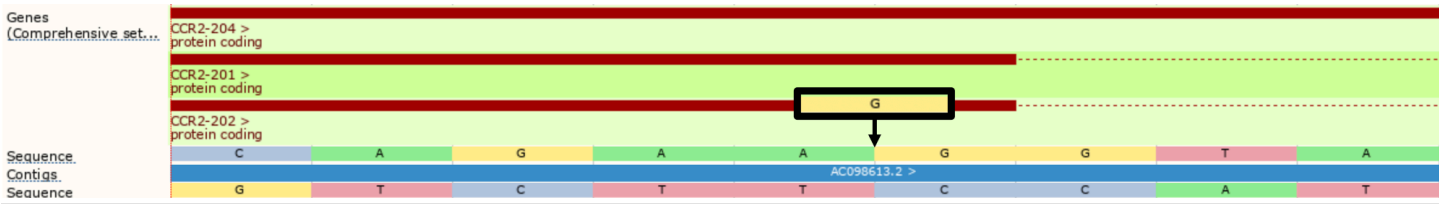
In addition to identifying known variants, VEP also reports allele frequencies for input alleles from major genotyping projects (1000 genomes, gnomAD exomes and gnomAD genomes). VEP's cache currently contains only frequency data for alleles that have been submitted to dbSNP or are imported via another 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_position        Amino_acids     Codons  Existing_variation
Extra
3_46358468_-/G  3:46358467-46358468     G       ENSG00000121807 ENST00000292301 Transcript
frameshift_variant,splice_region_variant        1425-1426       940-941 314     S/RX    agc/aGgc
-
       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
```

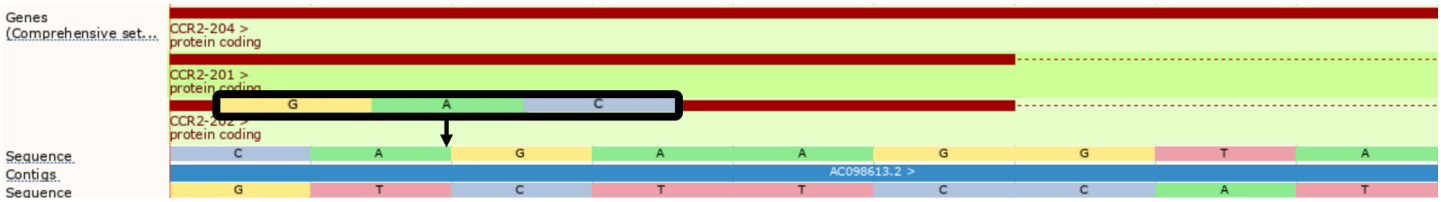
```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence
cDNA_position   CDS_position    Protein_position        Amino_acids     Codons  Existing_variation
Extra
3_46358468_-/G  3:46358467-46358468     G       ENSG00000121807 ENST00000292301 Transcript
splice_donor_variant    -       -       -       -       -       IMPACT=HIGH;STRAND=1
...
```

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_position        Amino_acids     Codons  Existing_variation
Extra
3_46358468_-/G  3:46358469-46358470     G       ENSG00000121807 ENST00000292301 Transcript
splice_donor_variant    -       -       -       -       -       -       IMPACT=HIGH;STRAND=1
...
```

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_position        Amino_acids     Codons  Existing_variation
Extra
3_46358465_-/GAC        3:46358464-46358465     CGA     ENSG00000121807 ENST00000292301 Transcript
inframe_insertion,splice_region_variant 1424-1425       939-940 313-314 -/R     -/CGA   -
IMPACT=MODERATE;STRAND=1
...
```

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

```
#Uploaded_variation     Location        Allele  Gene    Feature Feature_type    Consequence
cDNA_position   CDS_position    Protein_position        Amino_acids     Codons  Existing_variation
Extra
3_46358465_-/GAC        3:46358464-46358465     GAC     ENSG00000121807 ENST00000292301 Transcript
inframe_insertion       1422-1423       937-938 313     R/RR    aga/aGACga      -
IMPACT=MODERATE;STRAND=1
```

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). Also you can report issues through our (public) Github repositories. For general vep issues you should use ensembl-vep repository and for specific plugins you should use VEP_plugins repository.

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 your 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

Q: How to cite VEP?

If you use VEP, please cite our UPDATED publication so we can continue to support VEP development.

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.

Q: Is there a tutorial for web VEP?

Yes, see our latest tutorial Annotating and prioritizing genomic variants using the Ensembl Variant Effect Predictor — A tutorial for more information on using the Ensembl VEP web interface.

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 am I not seeing the same variant from my input in the output?

Since the Ensembl 110 release, VEP by default will minimise the input allele for display in the output. To see the exact allele representation you provided, use the --uploaded_allele option.

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

Q: Why do I get errors related with Sequence.pm?

```
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/Sequence.pm line 514.
Use of uninitialized value in concatenation (.) or string at /nfs/users/nfs_w/wm2/Perl/ensembl-
variation/modules/Bio/EnsEMBL/Variation/Utils/Sequence.pm line 643.
```

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 are chromosomes not found in annotation sources or synonyms?

```
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/114_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: Why do I get feature_type warnings from my GFF/GTF file?

```
WARNING: Ignoring 'five_prime_utr' feature_type from Homo_sapiens.GRCh38.111.gtf.gz GFF/GTF file.
This feature_type is not supported in VEP.
```

This can occur if you are using GFF/GTF file and the file contains a type that is not supported by VEP. Those lines are simply ignored. However, in cases where the transcript model is incomplete the full model may be ignored.

Please try to use supported feature types as mentioned here

Q: Can I get gnomAD exomes and genomes frequencies in VEP?

Yes, see this guide.

Q: Why do I have issues connecting to Ensembl databases?

```
Could not connect to database homo_sapiens_core_63_37 as user anonymous using
[DBI:mysql:database=homo_sapiens_core_63_37;host=ensembldb.ensembl.org;port=5306] as a locator:
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=ensembldb.ensembl.org;port=5306] as a locator:
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 use VEP with custom species and assemblies not available in Ensembl?

Yes - you can run VEP on any data you have by providing a custom GFF/GTF annotation and FASTA file, like so:

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

Q: Can I use VEP with T2T-CHM13 and other pangenome assemblies?

Yes - you can run VEP using Human Pangenome Reference Consortium (HPRC) data by following the instructions on how to use VEP with pangenomes assemblies.

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.

There are different approaches to download SIFT/PolyPhen data:

- Using the PolyPhen_SIFT plugin:
  - For any species with predictions in our Ensembl databases, the plugin is able to download the predictions data into a local SQLite database for offline use. PolyPhen predictions are only available for human data.
  - We also provide a downloadble SQLite database containing PolyPhen/SIFT predictions based on Human Pangenome Reference Consortium (HPRC) and GRCh38 assemblies. For more information, refer to Missense deleteriousness predictions in HPRC assemblies.
- Using our **Perl API**:
  - 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.
  - You would need to work out which peptide position your codon maps to, but there are methods in the `TranscriptVariation` class that should help you (probably `translation_start()` and `translation_end()`).