
Weblate Documentation

Release 3.7.1

Michal Čihař

Jun 28, 2019

1	Documentation	3
2	Installation	5
3	Bugs	7
4	License	9
	Python Module Index	307
	HTTP Routing Table	309
	Index	311

Weblate is a libre software web-based continuous localization system used by over 1150+ opensource projects & companies in over 115+ countries around the World.

The Weblate is to be found at <https://weblate.org/>

CHAPTER 1

Documentation

Detailed documentation is available in the `docs` directory of the sources.

The documentation can be also viewed online on <https://docs.weblate.org/>.

CHAPTER 2

Installation

Installation and setup instructions are provided in the manual, check the quick setup guide:

<https://docs.weblate.org/en/latest/admin/quick.html>

CHAPTER 3

Bugs

Please report bugs to <https://github.com/WeblateOrg/weblate/issues>

Copyright © 2012 - 2019 Michal Čihař michal@cihar.com

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

4.1 Weblate basics

4.1.1 Project structure

Internally, each project has translations to common strings propagated across other components within it by default. This lightens the burden of repetitive and multi version translation. Disable it as per *Component configuration*, still producing errors for seemingly inconsistent resulting translations.

4.2 Registration and user profile

4.2.1 Registration

While everybody can browse projects, view translations or suggest them, only registered users are allowed to actually save changes and are credited for every translation made.

You can register by following a few simple steps:

1. Fill out the registration form with your credentials

2. Activate registration by following in email you receive
3. Possibly adjust your profile to choose which languages you know

4.2.2 Dashboard

When you log in to Weblate, you will see an overview of projects and components as well as their translation progress. New in version 2.5.

By default, this will show the components of projects you are watching, cross-referenced with your preferred languages. You can switch to different views using the navigation tabs.

The screenshot shows the 'Your profile' page in Weblate. The top navigation bar includes 'Weblate', 'Dashboard', 'Watched projects', 'Projects', 'Your languages', and 'Languages'. Below this is a sub-navigation bar with 'Languages', 'Preferences' (selected), 'Notifications', 'Account', 'Profile', 'Licenses', 'Audit log', and 'API access'. The 'Preferences' section is titled 'Preferences' and contains several settings:

- ☐ Hide completed translations on the dashboard
- Translation editor mode**: A dropdown menu set to 'Full editor'.
- Zen editor mode**: A dropdown menu set to 'Top to bottom'.
- ☒ Show secondary translations in zen mode
- ☐ Hide source if there is secondary language
- Editor link**: A text input field with a placeholder text: 'Enter custom URL to be used as link to open source code. You can use {{branch}} for branch, {{filename}} and {{line}} as filename and line placeholders. Usually something like editor://open/?file={{filename}}&line={{line}} is good option.'
- Special characters**: A text input field with a placeholder text: 'You can specify additional special characters to be shown in the visual keyboard while translating. It can be useful for chars you use frequently but are hard to type on your keyboard.'
- Default dashboard view**: A radio button selection with options: 'Watched translations' (selected), 'Your languages', 'Component lists', 'Component list', and 'Suggested translations'.
- Default component list**: A dropdown menu set to '*****'.

At the bottom of the preferences section is a green 'Save' button. The footer of the page includes 'Powered by Weblate 3.7.1' and links to 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

The tabs will show several options:

- *All projects* will show translation status of all projects on the Weblate instance.
- *Your languages* will show translation status of all projects, filtered by your primary languages.
- *Watched* will show translation status of only those projects you are watching, filtered by your primary languages.

In addition, the drop-down can also show any number of *component lists*, sets of project components preconfigured by the Weblate administrator, see [Component Lists](#).

You can configure your preferred view in the *Preferences* section of your user profile settings.

4.2.3 User profile

User profile contains your preferences, name and email. Name and email are being used in VCS commits, so keep this information accurate.

Note: All language selections offers only languages which are currently being translated. If you want to translate to other language, please request it first on the project you want to translate.

Translated languages

Choose here which languages you prefer to translate. These will be offered to you on main page for watched projects to have easier access to these translations.

The screenshot shows the 'Your languages' page in the Weblate interface. At the top, there's a navigation bar with 'Weblate', 'Dashboard', 'Watched projects', 'Projects', 'Your languages', and 'Languages'. Below this is a 'Dashboard' section with tabs for 'Watched translations' (10), 'Your languages' (10), 'Suggested translations' (5), 'Insights', and 'Tools'. The main content is a table of watched translations.

Component	Strings	Words	Needs editing	Checks	Suggestions	Comments	
WeblateOrg/Android (Czech)	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Django (Czech)	96.2%	93.4%	0.0%	15.4%	0.0%	0.0%	Translate
WeblateOrg/Django (Hebrew)	92.3%	91.8%	0.0%	3.8%	0.0%	0.0%	Translate
WeblateOrg/Django (Hungarian)	69.2%	40.4%	11.5%	3.8%	0.0%	0.0%	Translate
WeblateOrg/Djangojs (Czech)	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Djangojs (Hebrew)	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Djangojs (Hungarian)	96.9%	94.5%	3.1%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Language names (Czech)	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Language names (Hebrew)	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
WeblateOrg/Language names (Hungarian)	81.8%	80.0%	13.6%	0.0%	0.0%	0.0%	Translate

Below the table, there are two buttons: 'Manage your languages' and 'Manage watched projects'. At the bottom, there's a footer with 'Powered by Weblate 3.7.1' and links for 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

Secondary languages

You can define secondary languages, which will be shown you while translating together with source language. Example can be seen on following image, where Slovak language is shown as secondary:

The screenshot displays the Weblate web interface for editing translations. The top navigation bar includes links for Dashboard, Watched projects, Projects, Your languages, and Languages. The breadcrumb trail shows the path: WeblateOrg / Django / Czech / translate. The main editing area is titled 'Translate' and shows the source text 'קבצים' and the translation 'Soubory' in Czech. A sidebar on the right contains a 'Glossary' section and 'Source information' including screenshot context, context, flags, source string location, source string age, translation file, and string priority. At the bottom, there is a table of 'Nearby strings' and a footer with project links.

Language	Status	Translation	Edit
Hebrew	✓	קבצים	Edit
Hungarian	✓	Fájlok	Edit

Default dashboard view

On the *Preferences* tab, you can pick which of the available dashboard views will be displayed by default. If you pick *Component list*, you have to select which component list will be displayed from the *Default component list* drop-down.

See also:

Component Lists

Avatar

Weblate can be configured to show avatar for each user (depending on `ENABLE_AVATARS`). These images are obtained using <https://gravatar.com/>.

Editor link

By default Weblate does display source code in web browser configured in the *Component configuration*. By setting *Editor link* you can override this to use your local editor to open the source code where translated strings is being used. You can use *Template markup*.

Usually something like `editor://open/?file={{filename}}&line={{line}}` is a good option.

See also:

You can find more information on registering custom URL protocols for editor in [nette documentation](#).

4.2.4 Notifications

You can subscribe to various notifications on *Subscriptions* tab. You will receive notifications for selected events on watched or administered projects.

Some of the notifications are sent only for events in your languages (for example about new strings to translate), while some trigger at component level (for example merge errors). These two groups of notifications are visually separated in the settings.

You can toggle notifications for watched projects, administered project and it can be further tweaked per project and component. To configure (or mute) notifications per project or component, visit component page and select appropriate choice from the *Watching* menu.

Note: You will not receive notifications for actions you've done.

[Dashboard](#)
[Watched projects](#)
[Projects](#)
[Your languages](#)
[Languages](#)

Your profile

[Languages](#)
[Preferences](#)
[Notifications](#)
[Account](#)
[Profile](#)
[Licenses](#)
[Audit log](#)
[API access](#)

Watched projects

Watched projects

Search...

Available:

WeblateOrg

Chosen:

WeblateOrg

You can receive notifications for watched projects and they are shown on the dashboard by default.
Watched projects are also shown on dashboard, so choose all projects you want to translate.

Save

Notification settings

Watched projects
Administered projects

Component wide notifications

You will receive a notification for every such event in your watched projects.

Merge failure
Do not notify

Parse error
Do not notify

Comment on authored translation
Instant notification

Mentioned in comment
Instant notification

New language
Do not notify

New translation component
Do not notify

New whiteboard message
Instant notification

New component alert
Do not notify

Translation notifications

You will only receive these notifications for your translated languages in your watched projects.

New string
Do not notify

New contributor
Do not notify

New suggestion
Do not notify

New comment
Do not notify

Changed string
Do not notify

Save

Powered by Weblate 3.7.1
[About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

4.2.5 Account

On the *Account* tab you can configure basic aspects of your account, connect various services which you can use to login into Weblate, completely remove your account or download your user data.

Note: List of services depends on Weblate configuration, but can include popular sites such as Google, Facebook, GitHub or Bitbucket.

Webplate

Dashboard

Projects

Your languages

Languages

+

?

Your profile

Languages

Preferences

Notifications

Account

Profile

Licenses

Audit log

API access

Account

Username

testuser

Username may only contain letters, numbers or the following characters: @ . + - _

Full name

Weblate Test

Email

weblate@example.org

You can add another email address below.

Your name and email will appear as author on the commits.

Save

Current user identities

Identity	User ID	Action
Password	testuser	<div>Change password</div>
Email	weblate@example.org	<div>Disconnect</div>
Google	weblate@example.org	<div>Disconnect</div>
GitHub	123456	<div>Disconnect</div>
Bitbucket	weblate	<div>Disconnect</div>

Add new association

Email

Removal

Removal of the account deletes all your private data.

Remove my account

User data

You can download all your private data.

Download user data

Powered by Weblate 3.7.1

About Weblate

Legal

Contact

Documentation

Donate to Weblate

4.3 Translating using Weblate

Thank you for interest in translating using Weblate. Projects can be either be set up for direct translation, or by way of accepting suggestions on behalf of users without accounts.

Overall, there are the modes of translation:

- Projects accepts direct translations
- Projects accepts only suggestions, which are accepted once given a defined number of votes

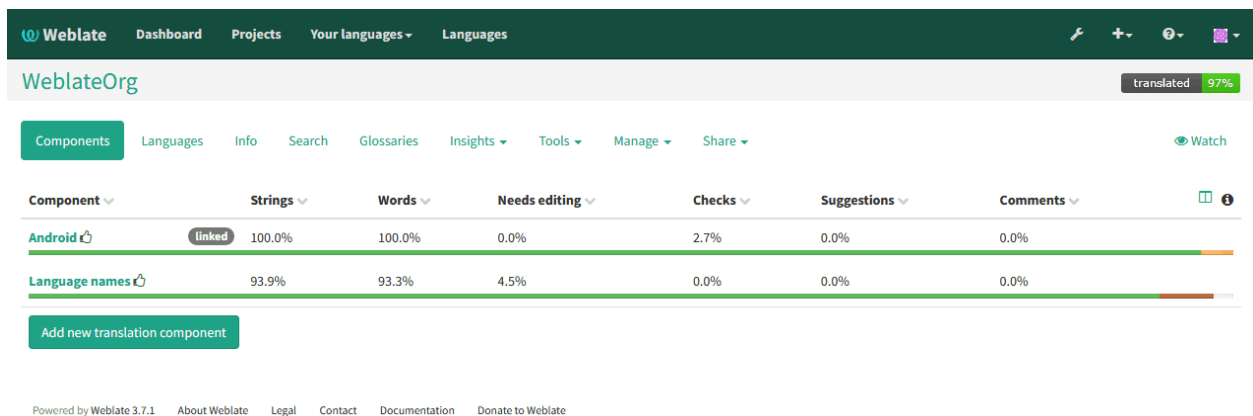
Options for translation project visibility:

- Publicly visible and anybody can contribute
- Visible only to a certain group of translators

Please see [Translation workflows](#) for more info on translation workflow.

4.3.1 Translation projects

Translation projects hold related components, related to the same software, book, or project.



4.3.2 Translation links

Having navigated to a component, a set of links lead to actual translation. The translation is further divided into individual checks, like “Untranslated” or “Needing review”. If the whole project is translated, without error, “All translations” is still available. Alternatively you can use the search field to find a specific string or term.

Weblate
Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Django / Czech

translated 96%

Overview
Info
Search
Glossary
Insights
Files
Tools
Manage
Share

Watching

Translation status

Strings

26

96.2%

Translate

Words

183

93.4%

Strings status

All strings

26

183 words

Translated strings

25

171 words

Strings needing action

1

12 words

Not translated strings

1

12 words

Strings needing action without suggestions

1

12 words

Strings with any failing checks

3

11 words

Source and translation are identical

1

4 words

Source and translation do not both end with a full stop

1

4 words

Python format string does not match source

1

3 words

Other components

Component	Strings	Words	Needs editing	Checks	Suggestions	Comments	
Language names	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
Android	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate
Djangojs	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%	Translate

Powered by Weblate 3.7.1
About Weblate
Legal
Contact
Documentation
Donate to Weblate

4.3.3 Suggestions

Note: Actual permissions might vary depending on your Weblate configuration.

Anonymous users can only (if permitted) forward suggestions. Doing so is still available to logged in users, in cases where uncertainty about the translation arises, which will prompt another translator to review it.

The suggestions are dialy scanned to remove duplicate ones or the one where suggestion matches current translation.

4.3.4 Translating

On the translation page, the source string and an edit area for translating it is shown. Should the translation be plural, multiple source strings and edit areas are shown, each described and label in plural form.

Any special whitespace chars you will find underlined in red and indicated with grey symbols. More than one subsequent space is also underlined in red to alert the translator to its formatting.

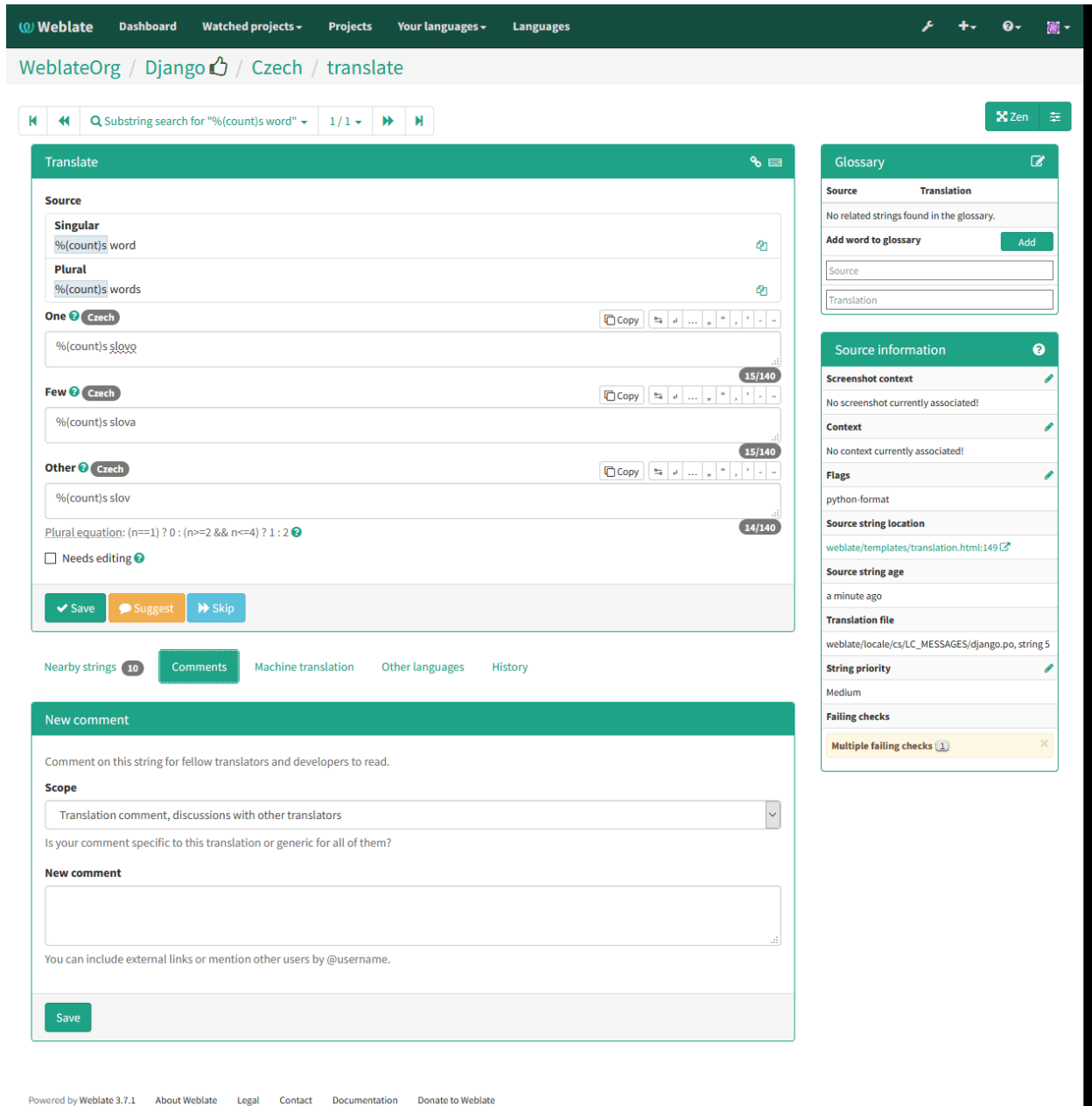
Various bits of extra info can be shown on this page, most which comes from the project source code (like context, comments or where the message is being used). When you choose secondary languages in your preferences, translation to these languages will be shown (see *Secondary languages*).

Below the translation, any suggestions made by others will be shown, which you can in turn accept, accept and make changes, or delete.

Plurals

Words that change form to account of their numeric designation are called plurals. Each language has its own definition of plurals. English, for example, supports one plural. In the singular definition of for example “car”, implicitly one car is referenced, in the plural definition, “cars” two or more cars are referenced, or the concept of cars as a noun. Languages like for example Czech or Arabic have more plurals and also their rules for plurals are different.

Weblate has full support for each of these forms, in each respective language by translating every plural separately. The number of fields and how it is used in the translated application depends on the configured plural equation. Weblate shows the basic info, but you can find a more detailed description in the [Language Plural Rules](#) by the Unicode Consortium.



The screenshot displays the Weblate web interface for translating a string. The top navigation bar includes links for Dashboard, Watched projects, Projects, Your languages, and Languages. The main content area is divided into several panels:

- Translate Panel:** Shows the source string '%(count)s word' and its plural forms in Czech. The 'One' form is '%(count)s slovo' and the 'Other' form is '%(count)s slov'. The plural equation is '(n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2'. There are buttons for 'Save', 'Suggest', and 'Skip'.
- Glossary Panel:** Shows no related strings found in the glossary. There is an 'Add word to glossary' button and input fields for 'Source' and 'Translation'.
- Source information Panel:** Shows details about the string, including its location ('weblate/templates/translation.html:149'), source string age ('a minute ago'), translation file ('weblate/locale/cs/LC_MESSAGES/django.po, string 5'), string priority ('Medium'), and failing checks ('Multiple failing checks').

At the bottom of the interface, there is a 'New comment' section for translators and developers to discuss the string. The footer includes links for 'Powered by Weblate 3.7.1', 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

Keyboard shortcuts

Changed in version 2.18: The keyboard shortcuts have been revamped in 2.18 to less likely collide with browser or system defaults.

The following keyboard shortcuts can be utilized during translation:

Alt+Home Navigates to first translation in current search.

Alt+End Navigates to last translation in current search.

Alt+PageUp Navigates to previous translation in current search.

Alt+PageDown Navigates to next translation in current search.

Ctrl+ or + or Ctrl+Enter or + Saves current translation.

Ctrl+Shift+ or +Shift+ Unmarks translation as fuzzy and submits it.

Ctrl+E or +E Focus translation editor.

Ctrl+U or +U Focus comment editor.

Ctrl+M or +M Shows machine translation tab.

Ctrl+<NUMBER> or +<NUMBER> Copies placeable of given number from source string.

Ctrl+M <NUMBER> or +M <NUMBER> Copy machine translation of given number to current translation.

Ctrl+I <NUMBER> or +I <NUMBER> Ignore failing check of given number.

Ctrl+J or +J Shows nearby strings tab.

Ctrl+S or +S Shows search tab.

Ctrl+O or +O Copies source string

Ctrl+T or +T Toggles edit needed flag.

Visual keyboard

A small visual keyboard is shown when translating. This can be useful for typing chars not usually found or otherwise hard to type.

The shown symbols factor into three categories:

- User configured chars defined in the *User profile*
- Per language chars provided by Weblate (e.g. quotes or RTL specific chars)
- Chars configured using *SPECIAL_CHARS*

Powered by Weblate 3.7.1 About Weblate Legal Contact Documentation Donate to Weblate

Translation context

This contextual description provides related info about the current string.

String attributes Things like message ID, context (msgctxt) or location in source code.

Screenshots Can be uploaded to Weblate to better inform translators of where and how the string is used, see [Visual context for strings](#).

Nearby messages Displays neighbouring messages from the translation file. These are usually also used in a similar context and prove useful in keeping the translation consistent.

Similar messages Messages found to be similar the current source string, which helps in providing a consistent translation.

All locations In case a message appears in multiple places (e.g. multiple components), this tab shows all of them if found to be inconsistent (see [Inconsistent](#)), you can choose which one to use.

Glossary Displays terms from the project glossary used in the current message.

Recent edits List of people whom have changed this message recently using Weblate.

Project Project info like instructions for translators, or info about its version control system repository.

If the translation format supports it, you can also follow supplied links to respective source code containing each source string.

Translation history

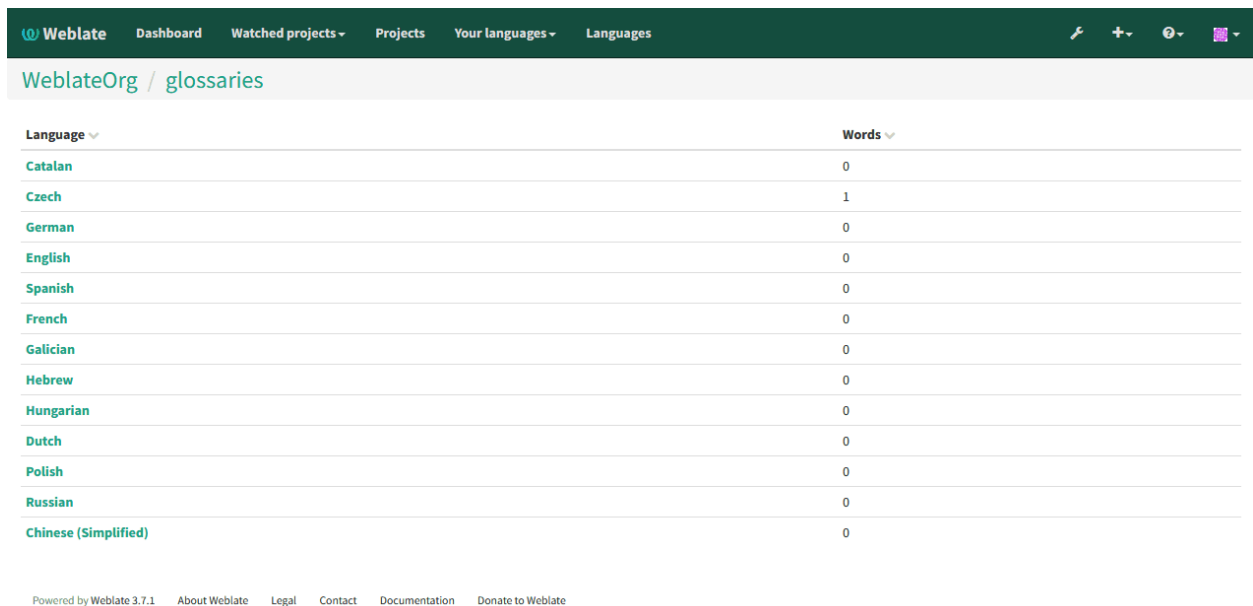
Every change is by default (unless turned off in component settings) saved in the database, and can be reverted. Optionally one can still also revert anything in the underlying version control system.

4.3.5 Glossary

Each project can have an assigned glossary for any language as a shorthand for storing terminology. Consistency is more easily maintained this way. Terms from the currently translated string can be displayed in the bottom tabs.

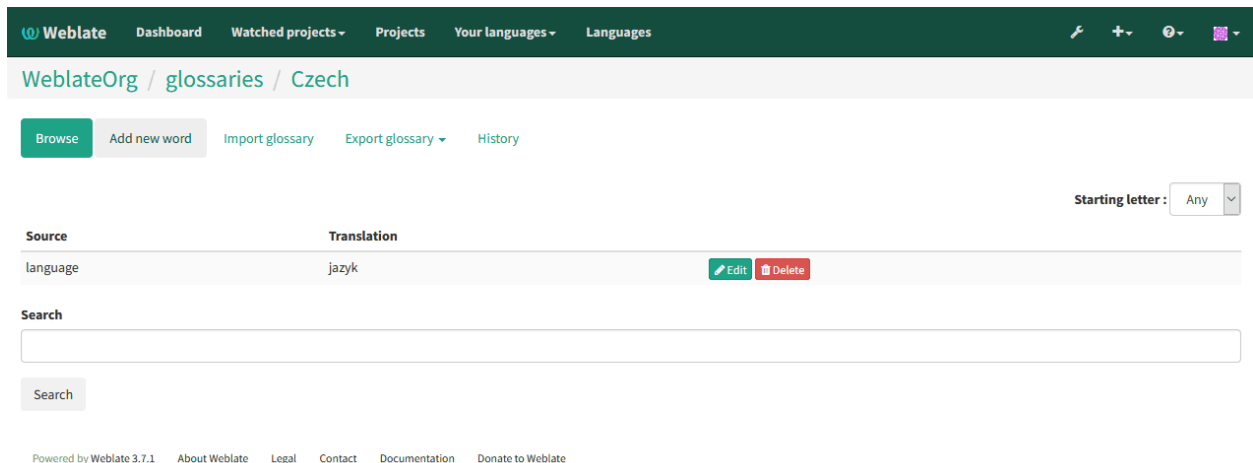
Managing glossaries

On the *Glossaries* tab of each project page, you can find a link that reads *Manage all glossaries*, wherein you can start new glossaries or edit existing ones. Once a glossary exists, it will also show up in this tab.



Language	Words
Catalan	0
Czech	1
German	0
English	0
Spanish	0
French	0
Galician	0
Hebrew	0
Hungarian	0
Dutch	0
Polish	0
Russian	0
Chinese (Simplified)	0

On the next page, you can choose which glossary to manage (all languages used in the current project are shown). Following this language link will lead you to a page which can be used to edit, import or export the glossary:



Starting letter: Any

Source	Translation
language	jazyk

Search

4.3.6 Machine translation

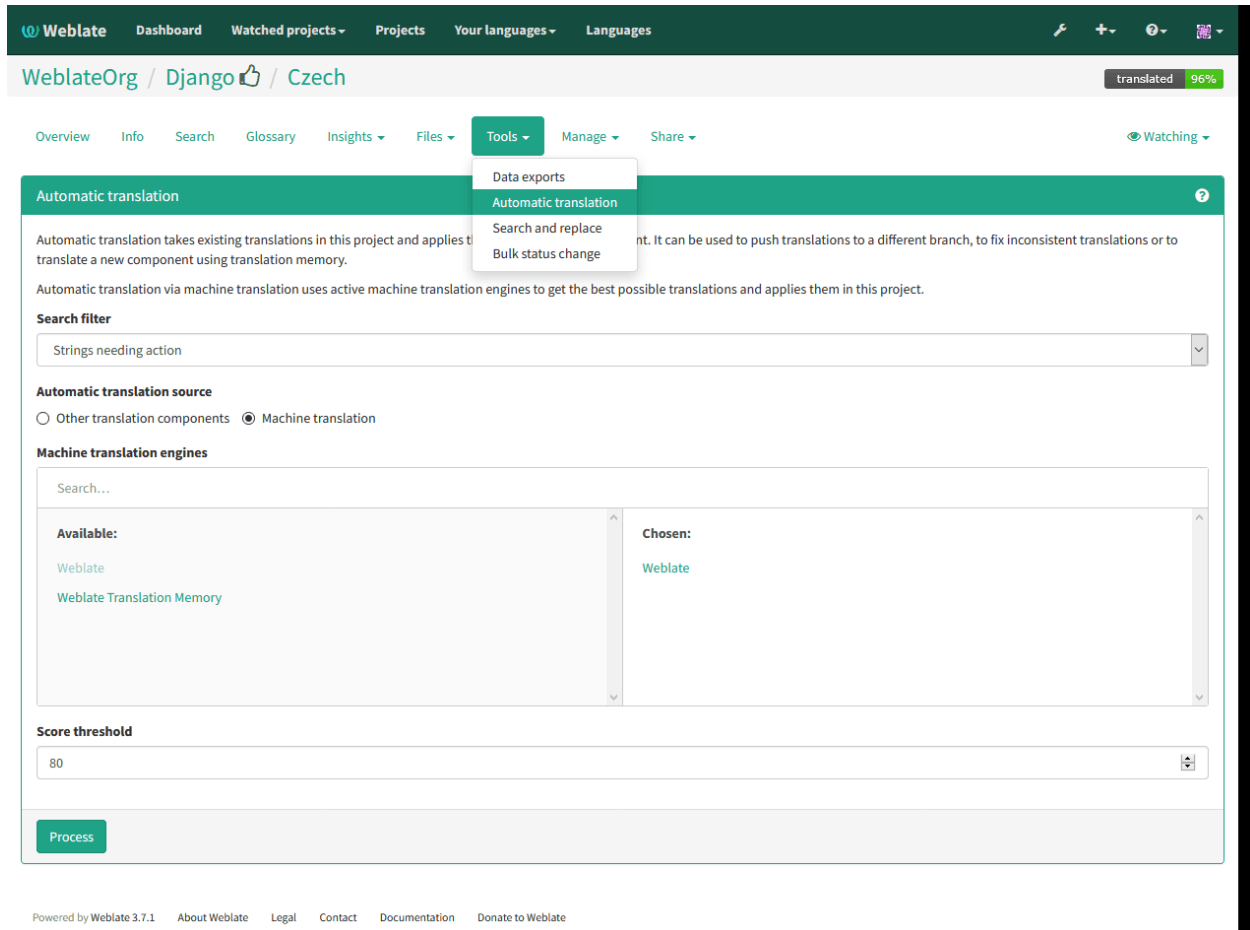
Based on configuration and your translated language, Weblate provides you suggestions from several machine translation tools. All machine translations are available in a single tab of each translation page.

See also:

You can find list of supported tools in *Machine translation*.

4.3.7 Automatic translation

You can use automatic translation to bootstrap translation based on external sources. This tool is called *Automatic translation* accessible in the *Tools* menu:



Two modes of operation are possible:

- Using other Weblate components as a source for translations.
- Using selected machine translation services with translations above a certain quality threshold.

You can also choose which strings are to be auto-translated.

Warning: Be mindful that this will overwrite existing translations if employed with wide filters such as *All strings*.

Useful in several situations like consolidating translation between different components (for example website and application) or when bootstrapping translation for a new component using existing translations (translation memory).

4.3.8 Rate limiting

To avoid abuse of the interface, there is rate limiting applied to several operations like searching, sending contact form or translating. In case you are hit by this, you are blocked for certain period until you can perform the operation again.

The default limits are described in the administrative manual in [Rate limiting](#), but can be tweaked by configuration.

4.4 Downloading and uploading translations

You can export files from a translation, make changes, and import them again. This allows working offline, and then merging changes back into the existing translation. This works even if it has been changed in the meantime.

Note: The available options might be limited by [Access control](#).

4.4.1 Downloading translations

Translatable files can be downloaded using the *Download source file* in the *Files* menu, producing a copy of the file as it is stored in upstream version control system.

Several file formats are available, including a compiled file to use in your choice of application (for example .mo files for GNU Gettext) using the *Files*.

4.4.2 Uploading translations

When you have made your changes, use *Upload translation* in the *Files* menu.

Any of supported file formats can be uploaded, but it is still recommended to use the same file format as is used for translation, otherwise some features might not be translated properly.

See also:

Supported file formats

The uploaded file is merged to update the translation, overwriting existing entries by default (this can be turned off or on in the upload dialog).

Import methods

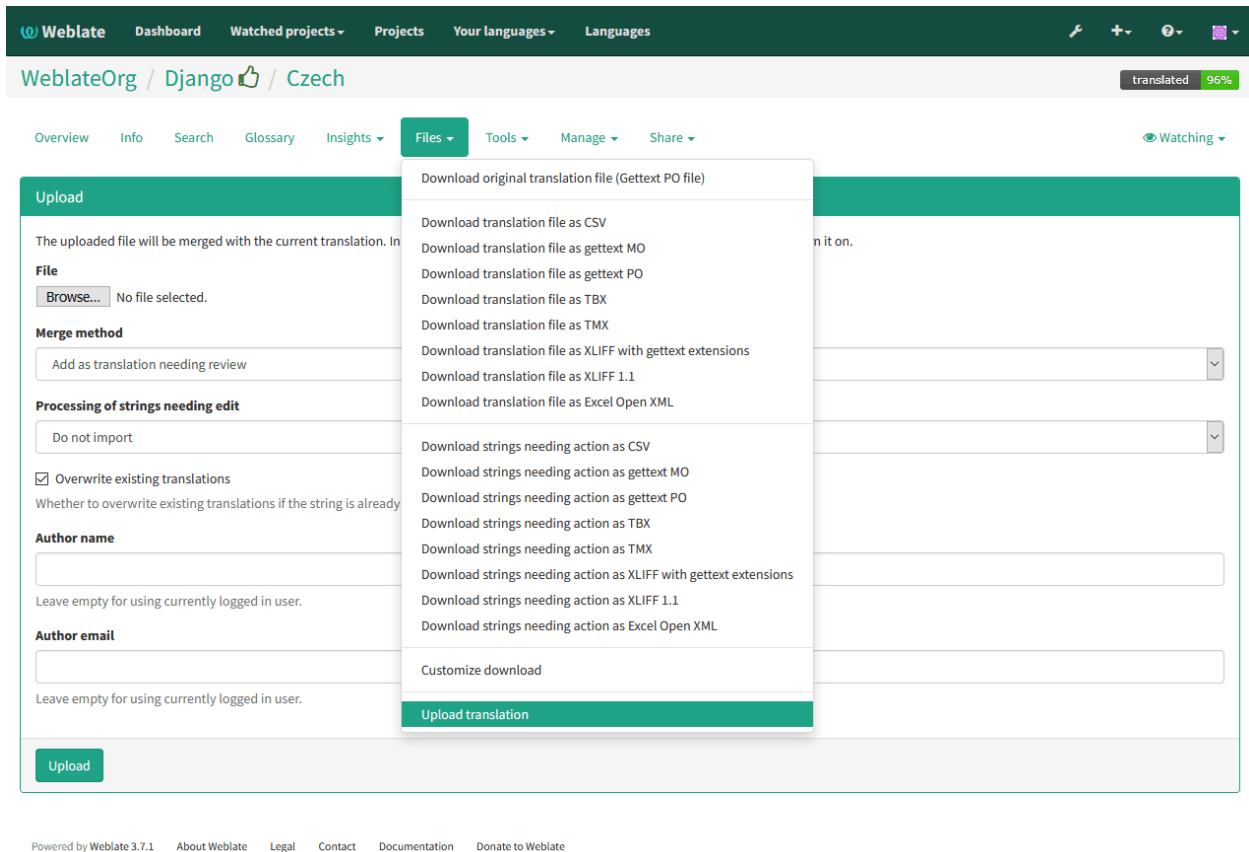
These are the choices presented when uploading translation files:

Add as translation Imported translations are added as translations. This is the most common usecase, and the default behavior.

Add as suggestion Imported translations are added as suggestions, do this when you want your uploaded strings reviewed.

Add as needing review Imported translations are added as translations needing review. This can be useful when you want translations to be used, but also reviewed.

There is also an option for how to handle strings needing review in the imported file.



4.5 Checks and fixups

The quality checks help catch common translator errors, ensuring the translation is in good shape. The checks are divided into three levels of severity, and can be ignored in case of false positives.

Once submitting a translation with a failing check, this is immediately shown to the user:

In addition to *Quality checks*, Weblate can also fix some common errors in translated strings automatically. Use it with caution to not have it add errors.

AUTOFIX_LIST

4.5.2 Quality checks

Weblate employs a wide range of quality checks on strings. The following section describes them all in further detail. There are also language specific checks. Please file a bug if anything is reported in error.

See also:

CHECK_LIST, *Customizing behavior*

4.5.3 Translation checks

Executed upon every translation change, helping translators maintain good quality translations.

Unchanged translation

Happens if the source and corresponding translation strings is identical, down to at least one of the plural forms. Some strings commonly found across all languages are ignored, and various markup is stripped. This reduces the number of false positives.

This check can help find strings mistakenly untranslated.

Starting or trailing newline

Source and translation do not both start (or end) with a newline.

Newlines usually appear in source strings for good reason, omissions or additions can lead to formatting problems when the translated text is put to use.

Starting spaces

Source and translation do not both start with the same number of spaces.

A space in the beginning of a string is usually used for indentation in the interface and thus important to keep.

Trailing space

Checks that trailing spaces are replicated between both source and translation.

Trailing space is usually utilized to space out neighbouring elements, so removing it might break layout.

Trailing stop

Checks that full stops are replicated between both source and translation. The presence of full stops is checked for various languages where they do not belong (Chinese, Japanese, Devanagari or Urdu).

Trailing colon

Checks that colons are replicated between both source and translation, and that they are correctly spaced. This includes rules for languages like French or Breton. The presence of colons is also checked for various languages where they do not belong (Chinese or Japanese).

Trailing question mark

Checks that question marks are replicated between both source and translation, and that they are correctly spaced or similar. This includes spacing rules for languages like French or Breton. The presence of question marks is also checked for various languages where they do not belong (Armenian, Arabic, Chinese, Korean, Japanese, Ethiopic, Vai or Coptic).

Trailing exclamation

Checks that exclamations are replicated between both source and translation, and that they are correctly spaced. This includes spacing rules for languages like French or Breton. The presence of exclamation marks is also checked for various languages where they do not belong (Chinese, Japanese, Korean, Armenian, Limbu, Myanmar or Nko).

Trailing ellipsis

Checks that trailing ellipses are replicated between both source and translation. This only checks for real ellipsis (. . .) not for three dots (. . .).

An ellipsis is usually rendered nicer than three dots in print, and sound better with text-to-speech.

See also:

[Ellipsis on wikipedia](#)

Trailing semicolon

Checks that semicolons at the end of sentences are replicated between both source and translation. This can be useful to keep formatting of entries such as desktop files.

Maximum Length

Checks that translations are of acceptable length to fit available space. This only checks for the length of translation characters.

Unlike the other checks, the flag should be set as a `key:value` pair like `max-length:100`.

Formatted strings

Checks that formatting in strings are replicated between both source and translation. Omitting format strings in translation usually cause severe problems, so the formatting in strings should usually match the source.

Weblate supports checking format strings in several languages. The check is not enabled automatically, only if a string is flagged appropriately (e.g. *c-format* for C format). Gettext adds this automatically, but you will probably have to add it manually for other file formats or if your PO files are not generated by **xgettext**.

This can be done per unit (see [Additional info on source strings](#)) or in [Component configuration](#). Having it defined per component is simpler, but can lead to false positives in case the string is not interpreted as a formatting string, but format string syntax happens to be used.

Besides checking, this will also highlight the formatting strings to easily insert them into translated strings:

Weblate Dashboard Watched projects ▾ Projects Your languages ▾ Languages

WeblateOrg / Django / Czech / translate

⏮ ⏪ 🔍 Substring search for "%(count)s word" ▾ 1 / 1 ▾ ⏩ ⏭

Zen ☰

Translate

Source

Singular
%(count)s word

Plural
%(count)s words

One Czech Copy 🔗 ⋮ ↶ ↷ ↺ ↻

%(count)s slovo

Few Czech Copy 🔗 ⋮ ↶ ↷ ↺ ↻

%(count)s slova

Other Czech Copy 🔗 ⋮ ↶ ↷ ↺ ↻

%(count)s slov

Plural equation: (n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2

☐ Needs editing

✓ Save 💬 Suggest ➡ Skip

Glossary

Source	Translation
No related strings found in the glossary.	
Add word to glossary Add	
Source	
Translation	

Source information

Screenshot context

No screenshot currently associated!

Context

No context currently associated!

Flags

python-format

Source string location

weblate/templates/translation.html:149

Source string age

a minute ago

Translation file

weblate/locale/cs/LC_MESSAGES/django.po, string 5

String priority

Medium

Failing checks

Multiple failing checks

Nearby strings 10 Comments Machine translation Other languages History

When	User	Action	Detail	Object
a minute ago	none	New source string		WeblateOrg/Django - Czech

Browse all component changes

Python format

Simple format string	There are %d apples
Named format string	Your balance is %(amount) %(currency)
Flag to enable	<i>python-format</i>

See also:

Python string formatting, Python Format Strings

Python brace format

Simple format string	There are {} apples
Named format string	Your balance is {amount} {currency}
Flag to enable	<i>python-brace-format</i>

See also:

Python brace format, Python Format Strings

PHP format

Simple format string	There are %d apples
Position format string	Your balance is %1\$d %2\$s
Flag to enable	<i>php-format</i>

See also:

[PHP sprintf documentation](#), [PHP Format Strings](#)

C format

Simple format string	There are %d apples
Position format string	Your balance is %1\$d %2\$s
Flag to enable	<i>c-format</i>

See also:

[C format strings](#), [C printf format](#)

Perl format

Simple format string	There are %d apples
Position format string	Your balance is %1\$d %2\$s
Flag to enable	<i>perl-format</i>

See also:

[Perl sprintf](#), [Perl Format Strings](#)

JavaScript format

Simple format string	There are %d apples
Flag to enable	<i>javascript-format</i>

See also:

[JavaScript formatting strings](#)

AngularJS interpolation string

Named format string	Your balance is {{amount}} {{ currency }}
Flag to enable	<i>angularjs-format</i>

See also:

[AngularJS: API: \\$interpolate](#)

C# format

Position format string	There are {0} apples
Flag to enable	<i>c-sharp-format</i>

See also:

[C# String Format](#)

Java format

Simple format string	There are %d apples
Position format string	Your balance is %1\$d %2\$s
Flag to enable	<i>java-format</i>

See also:

[Java Format Strings](#)

Java MessageFormat

Position format string	There are {0} apples
Flag to enable	<i>java-messageformat</i> enables the check unconditionally
	<i>auto-java-messageformat</i> enables check only if there is a format string in the source

See also:

[Java MessageFormat](#)

Missing plurals

Checks that all plural forms of a source string have been translated. Specifics on how each plural form is used can be found in the string definition.

Failing to fill in plural forms will in some cases lead to displaying nothing when the plural tense is in use.

Same plurals

Check that fails if some plural forms duplicated in the translation. In most languages they have to be different.

Inconsistent

Weblate checks translations of the same string across all translation within a project to help you keep consistent translations.

The check fails on differing translations of one string within a project. This can also lead to inconsistencies in displayed checks. You can find other translations of this string on the *All locations* tab.

Has been translated

Means a string has been translated already. This can happen when the translations have been reverted in VCS or lost otherwise.

Mismatched \n

Usually escaped newlines are important for formatting program output. Check fails if the number of `\\n` literals in translation do not match the source.

Mismatched BBCode

BBCode represents simple markup, like for example highlighting important parts of a message in bold font, or italics. This check ensures they are also found in translation.

Note: The method for detecting BBcode is currently quite simple so this check might produce false positives.

Zero-width space

Zero-width space (`<U+200B>`) character are used to truncate messages within words.

As they are usually inserted by mistake, this check is triggered once they are present in translation. Some programs might have problems when this character is used.

See also:

[Zero width space on Wikipedia](#)

Invalid XML markup

New in version 2.8.

The XML markup is not valid.

XML tags mismatch

This usually means the resulting output will look different. In most cases this is not desired result from changing the translation, but occasionally it is.

Checks that XML tags are replicated between both source and translation.

Markdown link references

New in version 3.5.

Markdown link references does not match source.

See also:

[Markdown links](#)

Markdown links

New in version 3.5.

Markdown links do not match source.

See also:

[Markdown links](#)

Markdown syntax

New in version 3.5.

Markdown syntax does not match source

See also:

[Markdown span elements](#)

Kashida letter used

New in version 3.5.

The decorative Kashida letters should not be used in translation. These are also known as Tatweel.

See also:

[Kashida on Wikipedia](#)

URL

New in version 3.5.

The translation does not contain an URL. This is triggered only in case the unit is marked as containing URL. In that case the translation has to be a valid URL.

Maximum size of translation

New in version 3.7.

Translation rendered text should not exceed given size. It renders the text with line wrapping and checks if it fits into given boundaries.

This check needs one or two parameters - maximal width and maximal number of lines. In case the number of lines is not provided, one line text is considered.

You can also configured used font by `font-*` directives (see [Customizing behavior](#)), for example following translation flags say that the text rendered with ubuntu font size 22 should fit into two lines and 500 pixels:

```
max-size:500:2, font-family:ubuntu, font-size:22
```

Hint: You might want to set `font-*` directives in [Component configuration](#) to have same font configured for all strings within a component. You can override those values per string in case you need to customize it per string.

See also:

Managing fonts, Customizing behavior

4.5.4 Source checks

Source checks can help developers improve the quality of source strings.

Optional plural

The string is optionally used as a plural, but does not use plural forms. In case your translation system supports this, you should use the plural aware variant of it.

For example with Gettext in Python it could be:

```
from gettext import ngettext
print ngettext('Selected %d file', 'Selected %d files', files) % files
```

Ellipsis

This fails when the string uses three dots (. . .) when it should use an ellipsis character (. . .).

Using the Unicode character is in most cases the better approach and looks better rendered, and may sound better with text-to-speech.

See also:

[Ellipsis on Wikipedia](#)

Multiple failing checks

Numerous translations of this string have failing quality checks. This is usually an indication that something could be done to improving the source string.

This check failing can quite often be caused by a missing full stop at the end of a sentence, or similar minor issues which translators tend to fix in translation, while it would be better to fix it in the source string.

4.6 Application developer guide

Using Weblate for translating your projects can bring you quite a lot of benefits. It's only up to you how much of that you will use.

4.6.1 Starting with internationalization

You have a project and want to to translate it into several languages? This guide will help you to do so. We will showcase several typical situations, but most of the examples are generic and can be applied to other scenarios as well.

Before translating any software, you should realize that languages around the world are really different and you should not make any assumption based on your experience. For most of languages it will look weird if you try to concatenate a sentence out of translated segments. You also should properly handle plural forms because many languages have complex rules for that and the internationalization framework you end up using should support this.

Last but not least, sometimes it might be necessary to add some context to the translated string. Imagine a translator would get string `Sun` to translate. Without context most people would translate that as our closest star, but it might be actually used as an abbreviation for Sunday.

Choosing internationalization framework

Choose whatever is standard on your platform, try to avoid reinventing the wheel by creating your own framework to handle localizations. Weblate supports most of the widely used frameworks, see [Supported file formats](#) for more information (especially [Translation types capabilities](#)).

Our personal recommendation for some platforms is in the following table. This is based on our experience, but that can not cover all use cases, so always consider your environment when doing the choice.

Platform	Recommended format
Android	Android string resources
iOS	Apple iOS strings
Qt	Qt Linguist .ts
Python	GNU Gettext
PHP	GNU Gettext ¹
C/C++	GNU Gettext
C#	.Net Resource files
Perl	GNU Gettext
Ruby	Ruby YAML files
Web extensions	WebExtension JSON
Java	XLIFF ²
JavaScript	JSON i18next files ³

Following chapters describe two use cases - GNU Gettext and Sphinx, but many of the steps are quite generic and apply to the other frameworks as well.

Translating software using GNU Gettext

[GNU Gettext](#) is one of the most widely used tool for internationalization of free software. It provides a simple yet flexible way to localize the software. It has great support for plurals, it can add further context to the translated string and there are quite a lot of tools built around it. Of course it has great support in Weblate (see [GNU Gettext](#) file format description).

Note: If you are about to use it in proprietary software, please consult licensing first, it might not be suitable for you.

GNU Gettext can be used from variety of languages (C, Python, PHP, Ruby, JavaScript and much more) and usually the UI frameworks already come with some support for it. The standard usage is though the `gettext()` function call, which is often aliased to `_()` to make the code simpler and easier to read.

Additionally it provides `pgettext()` call to provide additional context to translators and `ngettext()` which can handle plural types as defined for target language.

As a widely spread tool, it has many wrappers which make its usage really simple, instead of manual invoking of Gettext described below, you might want to try one of them, for example [intltool](#).

¹ The native Gettext support in PHP is buggy and often missing on Windows builds, it is recommended to use third party library [motranslator](#) instead.

² You can also use [Java properties](#) if plurals are not needed.

³ You can also use plain [JSON files](#) if plurals are not needed.

Sample program

The simple program in C using Gettext might look like following:

```
#include <libintl.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int count = 1;
    setlocale(LC_ALL, "");
    bindtextdomain("hello", "/usr/share/locale");
    textdomain("hello");
    printf(
        ngettext(
            "Orangutan has %d banana.\n",
            "Orangutan has %d bananas.\n",
            count
        ),
        count
    );
    printf("%s\n", gettext("Thank you for using Weblate."));
    exit(0);
}
```

Extracting translatable strings

Once you have code using the gettext calls, you can use **xgettext** to extract messages from it and store them into a .pot:

```
$ xgettext main.c -o po/hello.pot
```

Note: There are alternative programs to extract strings from the code, for example [pybabel](#).

This creates a template file, which you can use for starting new translations (using **msginit**) or updating existing ones after code change (you would use **msgmerge** for that). The resulting file is simply a structured text file:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2015-10-23 11:02+0200\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
```

(continues on next page)

(continued from previous page)

```
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\n"

#: main.c:14
#, c-format
msgid "Orangutan has %d banana.\n"
msgid_plural "Orangutan has %d bananas.\n"
msgstr[0] ""
msgstr[1] ""

#: main.c:20
msgid "Thank you for using Weblate."
msgstr ""
```

Each `msgid` line defines a string to translate, the special empty string in the beginning is the file header containing metadata about the translation.

Starting new translation

With the template in place, we can start our first translation:

```
$ msginit -i po/hello.pot -l cs --no-translator -o po/cs.po
Created cs.po.
```

The just created `cs.po` already has some information filled in. Most importantly it got the proper plural forms definition for chosen language and you can see number of plurals have changed according to that:

```
# Czech translations for PACKAGE package.
# Copyright (C) 2015 THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# Automatically generated, 2015.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2015-10-23 11:02+0200\n"
"PO-Revision-Date: 2015-10-23 11:02+0200\n"
"Last-Translator: Automatically generated\n"
"Language-Team: none\n"
"Language: cs\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=ASCII\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=3; plural=(n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2;\n"

#: main.c:14
#, c-format
msgid "Orangutan has %d banana.\n"
msgid_plural "Orangutan has %d bananas.\n"
msgstr[0] ""
msgstr[1] ""
msgstr[2] ""
```

(continues on next page)

(continued from previous page)

```
#: main.c:20
msgid "Thank you for using Weblate."
msgstr ""
```

This file is compiled into an optimized binary form, the `.mo` file used by the GNU `Gettext` functions at runtime.

Updating strings

Once you add more strings or change some strings in your program, you execute again `xgettext` which regenerates the template file:

```
$ xgettext main.c -o po/hello.pot
```

Then you can update individual translation files to match newly created templates (this includes reordering the strings to match new template):

```
$ msgmerge --previous --update po/cs.po po/hello.pot
```

Importing to Weblate

To import such translation into Weblate, all you need to define are the following fields when creating component (see *Component configuration* for detailed description of the fields):

Field	Value
Source code repository	URL of the VCS repository with your project
File mask	<code>po/*.po</code>
Base file for new translations	<code>po/hello.pot</code>
File format	Choose <i>Gettext PO file</i>
New language	Choose <i>Automatically add language file</i>

And that's it, you're now ready to start translating your software!

See also:

You can find a `Gettext` example with many languages in the Weblate Hello project on GitHub: <<https://github.com/WeblateOrg/hello>>.

Translating documentation using Sphinx

`Sphinx` is a tool for creating beautiful documentation. It uses simple `reStructuredText` syntax and can generate output in many formats. If you're looking for an example, this documentation is also build using it. The very useful companion for using `Sphinx` is the [Read the Docs](#) service, which will build and publish your documentation for free.

I will not focus on writing documentation itself, if you need guidance with that, just follow instructions on the [Sphinx](#) website. Once you have documentation ready, translating it is quite easy as `Sphinx` comes with support for this and it is quite nicely covered in their [Internationalization](#). It's matter of few configuration directives and invoking of the `sphinx-intl` tool.

If you are using [Read the Docs](#) service, you can start building translated documentation on the [Read the Docs](#). Their [Localization of Documentation](#) covers pretty much everything you need - creating another project, set its language and link it from master project as a translation.

Now all you need is translating the documentation content. As Sphinx splits the translation files per source file, you might end up with dozen of files, which might be challenging to import using the Weblate's web interface. For that reason, there is the `import_project` management command.

Depending on exact setup, importing of the translation might look like:

```
$ ./manage.py import_project --name-template 'Documentation: %s' \
  --file-format po \
  project https://github.com/project/docs.git master \
  'docs/locale/*/LC_MESSAGES/**/*.po'
```

If you have more complex document structure, importing different folders is not directly supported, you currently have to list them separately:

```
$ ./manage.py import_project --name-template 'Directory 1: %s' \
  --file-format po \
  project https://github.com/project/docs.git master \
  'docs/locale/*/LC_MESSAGES/dirl1/**/*.po'
$ ./manage.py import_project --name-template 'Directory 2: %s' \
  --file-format po \
  project https://github.com/project/docs.git master \
  'docs/locale/*/LC_MESSAGES/dir2/**/*.po'
```

See also:

The [Odorik](#) python module documentation is built using Sphinx, Read the Docs and translated using Weblate.

Integrating with Weblate

Getting translations updates from Weblate

To fetch updated strings from Weblate you can simply fetch the underlying repository (either from filesystem or it can be made available through [Git exporter](#)). Prior to this, you might want to commit any pending changes (see [Lazy commits](#)). This can be achieved in the user interface (in the *Repository maintenance*) or from command line using [Weblate Client](#).

This can be automated if you grant Weblate push access to your repository and configure *Push URL* in the [Component configuration](#).

See also:

[Continuous localization](#)

Pushing string changes to Weblate

To push newly updated strings to Weblate, just let it to pull from the upstream repo. This can be achieved in the user interface (in the *Repository maintenance*) or from command line using [Weblate Client](#).

This can be automated by installing a webhook on your repository to trigger Weblate whenever there is new commit, see [Updating repositories](#) for more details.

See also:

[Continuous localization](#)

4.6.2 Translation component alerts

Shows errors in the Weblate configuration or the translation project for any given translation component. Guidance on how to address found issues is also offered.

Currently the following is covered:

- Duplicated source strings in translation files
- Duplicated languages within translations
- Merge or update failures in the source repository
- Unused new base in component settings
- Parse errors in the translation files

Alerts are listed on each respective component page as *Alerts*. If it is missing, the component clears all current checks. Alerts can not be ignored, but will disappear once the underlying problem has been fixed.

A component with both duplicated strings and languages looks like this:

The screenshot shows the Weblate interface with the 'Alerts' tab selected. The page title is 'WeblateOrg / Duplicates'. A progress bar at the top right indicates 'translated 16%'. The 'Alerts' tab is highlighted in the navigation bar. Below the navigation bar, there are three alert boxes:

Duplicated translation.

The component contains several translation files mapped to single language in Weblate.
The following occurrences were found:

Language	Language codes
Czech	cs_CZ, cs

Please fix this by removing one of the duplicated strings from the translation files.

25 seconds ago

Duplicated string for translation.

The component contains several duplicated translation strings.
The following occurrences were found:

Language	Source
Italian	Thank you for using Weblate.

Please fix this by removing duplicated strings with same identifier from the translation files.

25 seconds ago

License information missing.

Any publicly available project should have defined license to indicate what terms apply to contributions.

25 seconds ago

At the bottom of the page, there is a footer with the text: 'Powered by Weblate 3.7.1' and links for 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

4.6.3 Managing translations

Adding new translations

Weblate can add new translations to your translation components when configured *Base file for new translations* (see [Component configuration](#)) or your file format doesn't require it (for most of monolingual files it is okay to start with blank files).

Weblate can be configured to automatically add translation when requested by user or to send notification to project admins for approval and manual processing. This can be done using *New translation* in [Component configuration](#). The project admins can still start translation within Weblate even if contact form is shown for regular users.

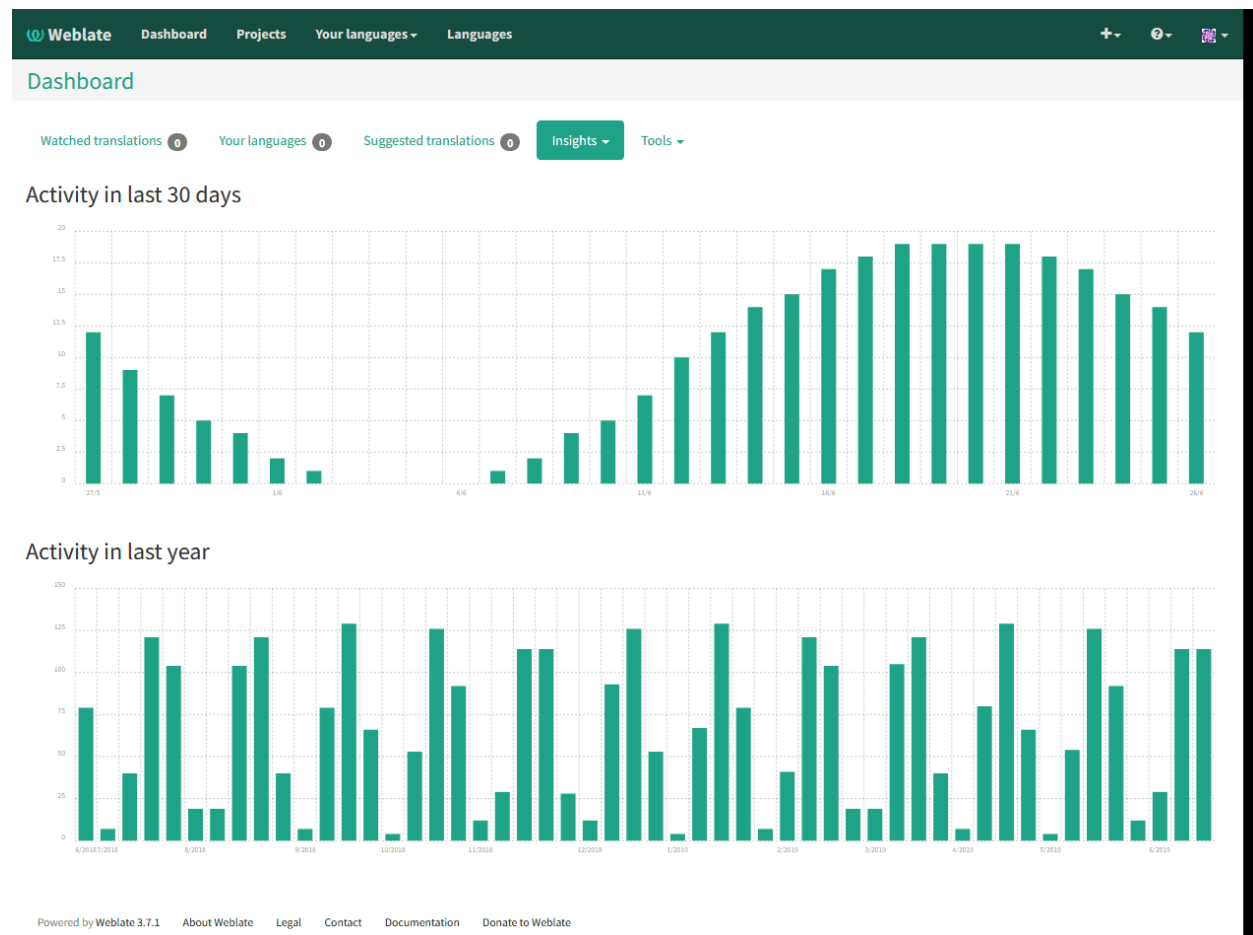
Alternatively you can add the files manually to the VCS. Weblate will automatically detect new languages which are added to the VCS repository and makes them available for translation. This makes adding new translations incredibly easy:

1. Add the translation file to VCS.
2. Let Weblate update the repository (usually set up automatically, see [Updating repositories](#)).

4.6.4 Reviewing source strings

Activity reports

Check changes of translations, projects or for individual users.



Source strings checks

There are many *Quality checks*, some of them focus on improving the quality of source strings. Many failing checks make for a hint to make source strings easier to translate. All types failing source checks are displayed on the *Source* tab of every component.

Translation string checks

Erroneous failing translation string checks indicate the problem is with the source string. Translators sometimes fix mistakes in the translation instead of reporting it - a typical example is a missing full stop at the end of a sentence.

Reviewing all failing checks of your translation, for every language, can provide valuable feedback to improve its source strings.

Source strings review is in the *Tools* menu of any given translation component. A similar view is presented when opening a translation, with slightly different checks displayed:

The screenshot displays the Weblate Django source strings view. The navigation bar at the top includes 'Weblate', 'Dashboard', 'Watched projects', 'Projects', 'Your languages', and 'Languages'. Below the navigation bar is a breadcrumb trail: 'WeblateOrg / Django / source strings'. The main content area is divided into two panels. The left panel, titled 'Strings status', shows a list of checks: 'All strings' (26), 'Strings with any failing source checks' (2), 'The string uses three dots (...) instead of an ellipsis character (...)' (1), and 'The translations in several languages have failing checks' (1). The right panel, titled 'Project Information', shows details about the project: 'Project website' (https://weblate.org/), 'Mailing list for translators' (weblate@lists.cihar.com), 'Instructions for translators' (https://weblate.org/contribute/), 'Translation process' (a list of bullet points), 'Translation license' (GPL-3.0+), 'Repository' (https://github.com/WeblateOrg/demo.git), 'Repository branch' (master 5b6ec48, a year ago), 'Repository containing Weblate translations' (http://localhost:9090/git/weblateorg/language-names/), and 'Filemask' (weblate/locale/*/LC_MESSAGES/django.po). At the bottom of the page, there is a footer with links: 'Powered by Weblate 3.7.1', 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

One of the most interesting checks here is the *Multiple failing checks* - it is triggered whenever there is failure on multiple translations of a given string. Usually this is something to look for, as this is a string translators have problems translating properly.

The detailed listing is a per language overview:

Weblate
Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Django / source strings / review

Source

Files

Priority

Medium

Failing checks

Language	Status	Checks	Edit
Czech	✓		Edit
Hebrew	✓		Edit
Hungarian	✓		Edit

Translation flags

Add

Please enter a comma separated list of translation flags, see [documentation](#) for more details.

Save

Priority

Medium

Higher priority strings are presented to translators earlier.

Save

Additional context

Save

Screenshot context

No screenshot currently associated! [Manage screenshots](#)

Add new screenshot

Screenshot name

Image

Browse...

No file selected.

Upload JPEG or PNG images up to 2000x2000 pixels.

Upload

Source string location

[weblate/templates/translation.html:45](#)
[weblate/trans/forms.py:1404](#)

Source string age

3 minutes ago

Translation file

weblate/locale/cs/LC_MESSAGES/django.po, string 1

Powered by Weblate 3.7.1

[About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

String comments

Translators can comment on both translation and source strings. Each *Component configuration* can be configured to receive such comments to an email address, and using the developers mailing list is usually the best approach. This way you can keep an eye on when problems arise in translation, tend to, and fix them quickly.

4.6.5 Promoting the translation

Weblate provides you widgets to share on your website or other sources to promote the translation project. It also has a nice welcome page for new contributors to give them basic information about the translation. Additionally you can share information about translation using Facebook or Twitter. All these possibilities can be found on the *Share* tab:

Promoting specific translations

Image widgets

Image svg

translated 97%

```
<a href="http://localhost:9090/engage/weblateorg/?utm_source=widget">

</a>
```

Color variants:



```
<a href="http://localhost:9090/engage/weblateorg/?utm_source=widget">

</a>

<a href="http://localhost:9090/engage/weblateorg/?utm_source=widget">

</a>

<a href="http://localhost:9090/engage/weblateorg/?utm_source=widget">

</a>

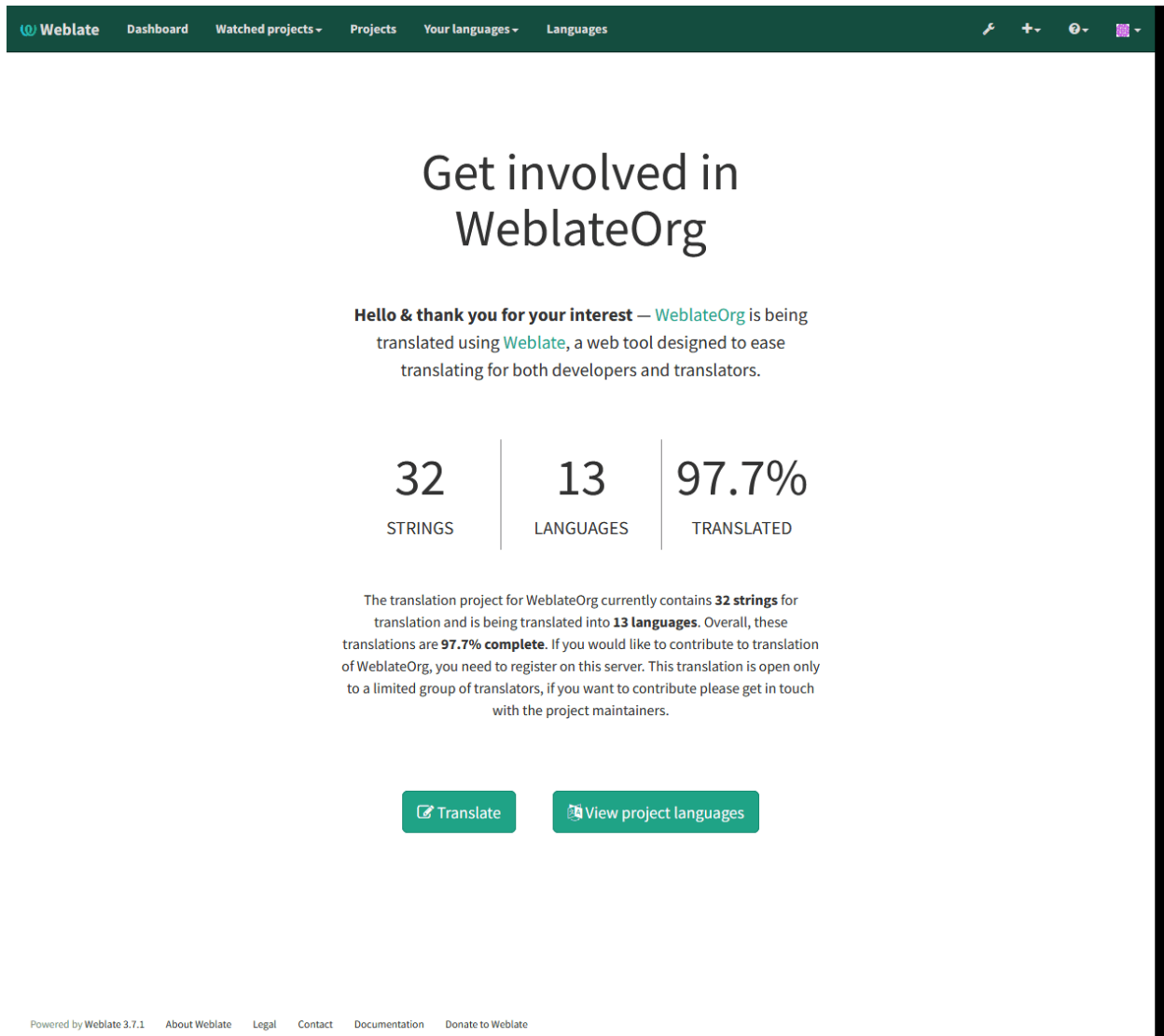
<a href="http://localhost:9090/engage/weblateorg/?utm_source=widget">

</a>
```

Color variants:



All these badges are provided with the link to simple page which explains users how to translate using Weblate:



The screenshot shows the Weblate web interface. At the top is a dark green navigation bar with the Weblate logo and links to Dashboard, Watched projects, Projects, Your languages, and Languages. Below the navigation bar, the main content area has a large heading "Get involved in WeblateOrg". Underneath, a message says "Hello & thank you for your interest — WeblateOrg is being translated using Weblate, a web tool designed to ease translating for both developers and translators." Below this, three statistics are displayed: 32 STRINGS, 13 LANGUAGES, and 97.7% TRANSLATED. A paragraph of text explains the translation project and provides instructions for contributors. At the bottom of the main content area, there are two green buttons: "Translate" and "View project languages". The footer of the page contains the text "Powered by Weblate 3.7.1" and links to About Weblate, Legal, Contact, Documentation, and Donate to Weblate.

4.6.6 Translation progress reporting

Reporting features give insight into how a translation progresses over a given period. A summary of contributions to any given component over time is provided. The reporting tool is found in the *Insights* menu of any translation component, project or on the dashboard:

Several reporting tools are available on this page and all can produce output in HTML, reStructuredText or JSON. The first two formats are suitable for embedding statistics into existing documentation, while JSON is useful for further processing of the data.

Translator credits

Generates a document usable for crediting translators - sorted by language and lists all contributors to a given language:

```
* Czech
    * Michal Čihař <michal@cihar.com>
    * John Doe <john@example.com>
* Dutch
    * Jane Doe <jane@example.com>
```

It will render as:

- Czech
 - Michal Čihař <michal@cihar.com>
 - John Doe <john@example.com>
- Dutch
 - Jae Doe <jane@example.com>

Contributor stats

Generates the number of translated words and strings by translator name:

4.6. Application developer guide

Name	Email	Count	Source words	Source chars	Target words	Target chars	Count	Source words	Source chars	Target words	Target chars	Count	Source words	Source chars	Target words	Target chars	Count	Source words	Source chars	Target words	Target chars
		total	total	total	total	total		new	new	new	new		approved	approved	approved	approved		edited	edited	edited	edited
Michal Čihař	malich@redhat.com	3	24	24	3	21	1	3	24	3	21	0	0	0	0	0	0	0	0	0	0
Alan Nordhøy	al-lan@example.com	2	5	25	4	28	2	3	24	3	21	0	0	0	0	0	0	0	0	0	0

It can be useful if you pay your translators based on amount of work, it gives you various stats on translators work.

All stats are available in three variants:

Total Overall number of edited strings.

New Newly translated strings which didn't have translation before.

Approved Count for string approvals in review workflow (see [Dedicated reviewers](#)).

Edited Edited strings which had translation before.

The following metrics are available for each:

Count Number of strings.

Source words Number of words in the source string.

Source chars Number of chars in the source string.

Target words Number of words in the translated string.

Target chars Number of chars in the translated string.

4.7 Translation workflows

Weblate can be configured to support several translation workflows. This document is not a complete listing of ways to configure Weblate, there are certainly more options. You can base another workflows on the most usual examples listed here.

4.7.1 Translation access

The [Access control](#) is not much discussed in the workflows as each of access control options can be applied to any workflows. Please consult that documentation for information how to manage access to translations.

In following chapters, *any user* means any user who has access to the translation. It can be any authenticated user if project is public or user having *Translate* permission on the project.

4.7.2 Translation states

Each translated string can be in following states:

Untranslated Translation is empty, it might or not be stored in the file, depending on the file format.

Needs editing Translation needs editing, this is usually result of source string change. The translation is stored in the file, depending on the file format it might be marked as needing edit (eg. fuzzy flag).

Waiting for review Translation is done, but not reviewed. It is stored in the file as a valid translation.

Approved Translation has been approved in the review. It can no longer be changed by translators, but only by reviewers. Translators can only add suggestions to it.

Suggestions Suggestions are stored in Weblate only and not in the translation file.

4.7.3 Direct translation

This is most usual setup for smaller teams - anybody can directly translate. This is also default setup in Weblate.

- *Any user* can edit translations.
- Suggestions are optional way to suggest changes, when translators are not sure about the change.

Setting	Value	Note
Enable reviews	disabled	configured at project level
Enable suggestions	enabled	it is useful for users to be able suggest when they are not sure
Suggestion voting	disabled	
Autoaccept suggestions	0	
Translators group	Users	or Translate with access control
Reviewers group	N/A	not used

4.7.4 Peer review

With this workflow, anybody can add suggestions, however they need approval from additional member before it is accepted as a translation.

- *Any user* can add suggestions
- *Any user* can vote for suggestions
- Suggestions become translations when they get given number of votes

Setting	Value	Note
Enable reviews	disabled	configured at project level
Enable suggestions	enabled	
Suggestion voting	enabled	
Autoaccept suggestions	1	you can set higher value to require more peer reviews
Translators group	Users	or Translate with access control
Reviewers group	N/A	not used, all translators review

4.7.5 Dedicated reviewers

New in version 2.18: The proper review workflow is supported since Weblate 2.18.

With dedicated reviewers you have two groups of users - one which can submit translations and one which reviews them. Review is there to ensure the translations are consistent and in a good quality.

- *Any user* can edit non approved translations.
- *Reviewer* can approve / unapproved strings.

- *Reviewer* can edit all translations (including approved ones).
- Suggestions are now also way to suggest changes for approved strings.

Setting	Value	Note
Enable reviews	enabled	configured at project level
Enable suggestions	enabled	it is useful for users to be able suggest when they are not sure
Suggestion voting	disabled	
Autoaccept suggestions	0	
Translators group	Users	or Translate with access control
Reviewers group	Reviewers	or Review with access control

4.7.6 Enabling reviews

The reviews can be enabled on project configuration, you can find the setting on bottom of *Manage users* page (to be found in the *Manage/Users* menu):

The screenshot shows the Weblate web interface. At the top is a navigation bar with links: Weblate, Dashboard, Watched projects, Projects, Your languages, and Languages. Below this is a breadcrumb trail: WeblateOrg / Manage users.

The main content area is titled 'Users' and contains a table of users. The table has columns for Username, Full name, Email, and several permission checkboxes: Administration, Billing, Glossary, Languages, Memory, Screenshots, Template, Translate, and VCS. A 'Remove' button is next to each user row. Below the table, a message states: 'The user will be removed from the project once all user permissions are removed.'

Below the user table is a section titled 'Add new user'. It includes a 'User to add' input field and a message: 'Please provide username or email. User needs to already have an active account in Weblate.' There is an 'Add' button at the bottom of this section.

Below the 'Add new user' section is a section titled 'Project access control'. It features a dropdown menu for 'Access control' currently set to 'Protected'. Below this, there is explanatory text: 'How to restrict access to this project is detailed in the documentation.' There are three radio button options: 'Public' (Publicly visible and translatable), 'Protected' (Publicly visible, only translatable for chosen users), and 'Private' (Visible and translatable only for chosen users). There is also a 'Custom' option with a warning: 'Only use this if you know what you are doing, enabling it might revoke your access to this project. Permissions are not managed in Weblate.' Below these options is a checkbox for 'Enable reviews' with the text: 'Requires dedicated reviewers to approve translations.'

At the bottom of the 'Project access control' section, a message states: 'You do not have permission to change project access control.' There is a 'Check your billing status' button.

The footer of the page includes the text: 'Powered by Weblate 3.7.1' followed by links: About Weblate, Legal, Contact, Documentation, and Donate to Weblate.

Note: Depending on Weblate configuration, the setting might not be available to you. For example on Hosted Weblate this is not available for projects hosted for free.

4.8 Frequently Asked Questions

4.8.1 Configuration

How to create an automated workflow?

Weblate can handle all the translation things semi-automatically for you. If you give it push access to your repository, the translations can happen without interaction unless some merge conflict occurs.

1. Set up your git repository to tell Weblate whenever there is any change, see *Notification hooks* for information how to do it.
2. Set push URL at your *Component configuration* in Weblate, this will allow Weblate to push changes to your repository.
3. Enable push on commit on your *Project configuration* in Weblate, this will make Weblate push changes to your repository whenever they are committed at Weblate.

See also:

Continuous localization, Avoiding merge conflicts

How to access repositories over SSH?

Please see *Accessing repositories* for information about setting up SSH keys.

How to fix merge conflicts in translations?

The merge conflicts happen from time to time when the translation file is changed in both Weblate and the upstream repository. You can usually avoid this by merging Weblate translations prior to doing some changes in the translation files (e.g. before executing msgmerge). Just tell Weblate to commit all pending translations (you can do it in the *Repository maintenance* in the *Tools* menu) and merge the repository (if automatic push is not enabled).

If you've already ran into the merge conflict, the easiest way is to solve all conflicts locally at your workstation - simply add Weblate as a remote repository, merge it into upstream and fix any conflicts. Once you push changes back, Weblate will be able to use the merged version without any other special actions.

Note: Depending on your setup, access to the Weblate repository might require authentication. When using Weblate built in *Git exporter*, you authenticate with your username and the API key.

```
# Commit all pending changes in Weblate, you can do this in the UI as well
wlc commit
# Lock translation in Weblate, again this can be done in the UI as well
wlc lock
# Add Weblate as remote
git remote add weblate https://hosted.weblate.org/git/project/component/
# You might need to include credentials in some cases:
```

(continues on next page)

(continued from previous page)

```
git remote add weblate https://username:APIKEY@hosted.weblate.org/git/project/
↪component/

# Update weblate remote
git remote update weblate

# Merge Weblate changes
git merge weblate/master

# Resolve conflicts
edit ....
git add ...
...
git commit

# Push changes to upstream repository, Weblate will fetch merge from there
git push

# Open Weblate for translation
wlc unlock
```

If you're using multiple branches in Weblate, you can work similarly on all branches:

```
# Add and update Weblate remotes
git remote add weblate-one https://hosted.weblate.org/git/project/one/
git remote add weblate-second https://hosted.weblate.org/git/project/second/
git remote update weblate-one weblate-second

# Merge QA_4_7 branch
git checkout QA_4_7
git merge weblate-one/QA_4_7
... # Resolve conflicts
git commit

# Merge master branch
git checkout master
git merge weblate-second/master
... # Resolve conflicts
git commit

# Push changes to upstream repository, Weblate will fetch merge from there
git push
```

In case of Gettext po files, there is a way to merge conflict in a semi-automatic way:

Get and keep local clone of the Weblate git repository. Also get a second fresh local clone of the upstream git repository (i. e. you need two copies of the upstream git repository: intact and working copy):

```
# Add remote
git remote add weblate /path/to/weblate/snapshot/

# Update weblate remote
git remote update weblate

# Merge Weblate changes
git merge weblate/master
```

(continues on next page)

(continued from previous page)

```
# Resolve conflicts in the po files
for PO in `find . -name '*.po'` ; do
    msgcat --use-first /path/to/weblate/snapshot/$PO\
        /path/to/upstream/snapshot/$PO -o $PO.merge
    msgmerge --previous --lang=${PO%.po} $PO.merge domain.pot -o $PO
    rm $PO.merge
    git add $PO
done
git commit

# Push changes to upstream repository, Weblate will fetch merge from there
git push
```

See also:

How to export the Git repository that Weblate uses?, Continuous localization, Avoiding merge conflicts

How do I translate several branches at once?

Weblate supports pushing translation changes within one *Project configuration*. For every *Component configuration* which has it enabled (the default behavior), the change made is automatically propagated to others. This way the translations are kept synchronized even if the branches themselves have already diverged quite a lot and it is not possible to simply merge translation changes between them.

Once you merge changes from Weblate, you might have to merge these branches (depending on your development workflow) discarding differences:

```
git merge -s ours origin/maintenance
```

How to export the Git repository that Weblate uses?

There is nothing special about the repository, it lives under the `DATA_DIR` directory and is named `vcs/<project>/<component>/`. If you have SSH access to this machine, you can use the repository directly.

For anonymous access you might want to run a git server and let it serve the repository to the outside world.

Alternatively you can use *Git exporter* inside Weblate to automate this.

What are the options for pushing changes back upstream?

This heavily depends on your setup, Weblate is quite flexible in this area. Here are examples of workflows used with Weblate:

- Weblate automatically pushes and merges changes (see *How to create an automated workflow?*)
- You manually tell Weblate to push (it needs push access to the upstream repository)
- Somebody manually merges changes from the Weblate git repository into the upstream repository
- Somebody rewrites history produced by Weblate (eg. by eliminating merge commits), merges changes and tells Weblate to reset the content on the upstream repository.

Of course you are free to mix all of these as you wish.

How can I limit Weblate access to translations only without exposing source code to it?

You can use `git submodule` for separating translations from source code while still having them under version control.

1. Create a repository with your translation files.
2. Add this as a submodule to your code:

```
git submodule add git@example.com:project-translations.git path/to/translations
```

3. Link Weblate to this repository, it no longer needs access to the repository with your source code.
4. You can update the main repository with translations from Weblate by:

```
git submodule update --remote path/to/translations
```

Please consult `git submodule` documentation for more details.

How can I check if my Weblate is configured properly?

Weblate includes a set of configuration checks which you can see in the admin interface, just follow the *Performance report* link in the admin interface or open the `/admin/performance/` URL directly.

Why do links contain `example.com` as the domain?

Weblate uses Django's sites framework and it defines the site name inside the database. You need to set the domain name to match your installation.

See also:

Set correct sitename

Why are all commits committed by Weblate <noreply@weblate.org>?

This is the default committer name, configured when you create a translation component. You can also change it in the administration at any time.

The author of every commit (if the underlying VCS supports it) is still recorded correctly as the user who has made the translation.

See also:

Component configuration

4.8.2 Usage

How do I review others translations?

- You can subscribe to any changes made in *Notifications* and then check others contributions in email.
- There is a review tool available at the bottom of the translation view, where you can choose to browse translations made by others since a given date.

How do I provide feedback on a source string?

On context tabs below translation, you can use the *Source* tab to provide feedback on a source string or discuss it with other translators.

How can I use existing translations while translating?

Weblate provides you with several ways to utilize existing translations while translating:

- You can use the import functionality to load compendium as translations, suggestions or translations needing review. This is the best approach for a one time translation using compendium or similar translation database.
- You can setup *tmserver* with all databases you have and let Weblate use it. This is good for cases when you want to use it for several times during translating.
- Another option is to translate all related projects in a single Weblate instance, which will make it automatically pick up translations from other projects as well.

See also:

Machine translation, Machine translation

Does Weblate update translation files besides translations?

Weblate tries to limit changes in translation files to a minimum. For some file formats it might unfortunately lead to reformatting the file. If you want to keep the file formatted in your way, please use a pre-commit hook for that.

For monolingual files (see *Supported file formats*) Weblate might add new translation strings which are present in the *template* and not in actual translations. It does not however perform any automatic cleanup of stale strings as that might have unexpected outcomes. If you want to do this, please install a pre-commit hook which will handle the cleanup according to your requirements.

Weblate also will not try to update bilingual files in any way, so if you need `po` files being updated from `pot`, you need to do it yourself.

See also:

Processing repository with scripts

Where do language definitions come from and how can I add my own?

The basic set of language definitions is included within Weblate and Translate-toolkit. This covers more than 150 languages and includes information about used plural forms or text direction.

You are free to define own languages in the administrative interface, you just need to provide information about it.

Can Weblate highlight changes in a fuzzy string?

Weblate supports this, however it needs the data to show the difference.

For Gettext PO files, you have to pass the parameter `--previous` to **msgmerge** when updating PO files, for example:

```
msgmerge --previous -U po/cs.po po/phpmyadmin.pot
```

For monolingual translations, Weblate can find the previous string by ID, so it shows the differences automatically.

Why does Weblate still show old translation strings when I've updated the template?

Weblate does not try to manipulate the translation files in any way other than allowing translators to translate. So it also does not update the translatable files when the template or source code have been changed. You simply have to do this manually and push changes to the repository, Weblate will then pick up the changes automatically.

Note: It is usually a good idea to merge changes done in Weblate before updating translation files, as otherwise you will usually end up with some conflicts to merge.

For example with Gettext PO files, you can update the translation files using the **msgmerge** tool:

```
msgmerge -U locale/cs/LC_MESSAGES/django.mo locale/django.pot
```

In case you want to do the update automatically, you can install add-on *Update PO files to match POT (msgmerge)*.

4.8.3 Troubleshooting

Requests sometimes fail with too many open files error

This happens sometimes when your Git repository grows too much and you have many of them. Compressing the Git repositories will improve this situation.

The easiest way to do this is to run:

```
# Go to DATA_DIR directory
cd data/vcs
# Compress all Git repositories
for d in */* ; do
    pushd $d
    git gc
    popd
done
```

See also:

[DATA_DIR](#)

Fulltext search is too slow

Depending on various conditions (frequency of updates, server restarts and other), the fulltext index might become too fragmented over time. It is recommended to optimize it from time to time:

```
./manage.py rebuild_index --optimize
```

In case it does not help (or if you have removed a lot of strings) it might be better to rebuild it from scratch:

```
./manage.py rebuild_index --clean
```

See also:

[rebuild_index](#)

I get “Lock Error” quite often while translating

This is usually caused by concurrent updates to the fulltext index. In case you are running a multi-threaded server (e.g. `mod_wsgi`), this happens quite often. For such a setup it is recommended to use Celery to perform updates in the background.

See also:

Fulltext search, Background tasks using Celery

Rebuilding index has failed with “No space left on device”

Whoosh uses a temporary directory to build indices. In case you have a small `/tmp` (eg. using ramdisk), this might fail. Change the temporary directory by passing it as `TEMP` variable:

```
TEMP=/path/to/big/temp ./manage.py rebuild_index --clean
```

See also:

rebuild_index

Database operations fail with “too many SQL variables”

This can happen when using the SQLite database as it is not powerful enough for some relations used within Weblate. The only way to fix this is to use some more capable database, see *Use a powerful database engine* for more information.

See also:

Use a powerful database engine, Databases

When accessing the site I get Bad Request (400) error

This is most likely caused by an improperly configured `ALLOWED_HOSTS`. It needs to contain all hostnames you want to access your Weblate. For example:

```
ALLOWED_HOSTS = ['weblate.example.com', 'weblate', 'localhost']
```

See also:

Allowed hosts setup

4.8.4 Features

Does Weblate support other VCS than Git and Mercurial?

Weblate currently does not have native support for anything other than *Git* (with extended support for *GitHub*, *Gerrit* and *Subversion*) and `ref:vcs-mercurial`, but it is possible to write backends for other VCSes.

You can also use *Git remote helpers* in Git to access other VCSes.

Note: For native support of other VCS, Weblate requires distributed VCS and could be probably adjusted to work with anything other than Git and Mercurial, but somebody has to implement this support.

See also:

[Version control integration](#)

How does Weblate credit translators?

Every change made in Weblate is committed into VCS under the translators name. This way every single change has proper authorship and you can track it down using standard VCS tools you use for code.

Additionally, when the translation file format supports it, the file headers are updated to include the translator name.

See also:

[list_translators](#)

Why does Weblate force to show all po files in a single tree?

Weblate was designed in a way that every po file is represented as a single component. This is beneficial for translators, so they know what they are actually translating. If you feel your project should be translated as one, consider merging these po files. It will make life easier even for translators not using Weblate.

Note: In case there will be big demand for this feature, it might be implemented in future versions, but it's definitely not a priority for now.

Why does Weblate use language codes such sr_Latn or zh_Hant?

These are language codes defined by [RFC 4646](#) to better indicate that they are really different languages instead previously wrongly used modifiers (for @latin variants) or country codes (for Chinese).

Weblate will still understand legacy language codes and will map them to current one - for example sr@latin will be handled as sr_Latn or zh@CN as sr_Hans.

4.9 Supported file formats

Weblate supports most translation format understood by the translate-toolkit, however each format being slightly different, there might be some issues with formats that are not well tested.

See also:

[Translation Related File Formats](#)

Note: When choosing a file format for your application, it's better to stick some well established format in the toolkit/platform you use. This way your translators can use whatever tools they are get used to and will more likely contribute to your project.

4.9.1 Bilingual and monolingual formats

Weblate does support both monolingual and bilingual formats. Bilingual formats store two languages in single file - source and translation (typical examples are *GNU Gettext*, *XLIFF* or *Apple iOS strings*). On the other side, monolingual formats identify the string by ID and each language file contains only mapping of those to given language (typically *Android string resources*). Some file formats are used in both variants, see detailed description below.

For correct use of monolingual files, Weblate requires access to a file containing complete list of strings to translate with their source - this file is called *Monolingual base language file* within Weblate, though the naming might vary in your application.

4.9.2 Automatic detection

Weblate can automatically detect several widely spread file formats, but this detection can harm your performance and will limit features specific to given file format (for example automatic adding of new translations).

4.9.3 Translation types capabilities

Below are listed capabilities of all supported formats.

Format	Lingual-ity ¹	Plu-rals ²	Com-ments ³	Con-text ⁴	Loca-tion ⁵	Flags ⁸	Additional states ⁶
<i>GNU Gettext</i>	bilingual	yes	yes	yes	yes	yes ⁹	needs editing
<i>Monolingual Gettext</i>	mono	yes	yes	yes	yes	no	needs editing
<i>XLIFF</i>	both	yes	yes	yes	yes	par-tial ¹⁰	needs editing, ap-proved
<i>Java properties</i>	both	no	yes	no	no	no	
<i>Joomla translations</i>	mono	no	yes	no	yes	no	
<i>Qt Linguist .ts</i>	both	yes	yes	no	yes	no	needs editing
<i>Android string re-sources</i>	mono	yes	yes ⁷	no	no	no	
<i>Apple iOS strings</i>	bilingual	no	yes	no	no	no	
<i>PHP strings</i>	mono	no	yes	no	no	no	
<i>JSON files</i>	mono	no	no	no	no	no	
<i>JSON i18next files</i>	mono	yes	no	no	no	no	
<i>WebExtension JSON</i>	mono	yes	yes	no	no	no	
<i>.Net Resource files</i>	mono	no	yes	no	no	no	
<i>CSV files</i>	mono	no	yes	yes	yes	no	needs editing
<i>YAML files</i>	mono	no	yes	no	no	no	
<i>Ruby YAML files</i>	mono	yes	yes	no	no	no	
<i>DTD files</i>	mono	no	no	no	no	no	
<i>Windows RC files</i>	mono	no	yes	no	no	no	
<i>Excel Open XML</i>	mono	no	yes	yes	yes	no	needs editing
<i>App store metadata files</i>	mono	no	no	no	no	no	
<i>Subtitle files</i>	mono	no	no	no	yes	no	

¹ See *Bilingual and monolingual formats*

² Plurals are necessary to properly localize strings with variable count.

³ Comments can be used to pass additional information about string to translate.

⁴ Context is used to differentiate same strings used in different scope (eg. *Sum* can be used as abbreviated name of day or as a name of our closest

4.9.4 GNU Gettext

Most widely used format in translating free software. This was first format supported by Weblate and still has the best support.

Weblate supports contextual information stored in the file, adjusting its headers or linking to corresponding source files.

The bilingual gettext PO file typically looks like:

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "Monday"
msgstr "Pondělí"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "Tuesday"
msgstr "Úterý"

#: weblate/accounts/avatar.py:163
msgctxt "No known user"
msgid "None"
msgstr "Žádný"
```

Typical Weblate <i>Component configuration</i>	
File mask	po/* .po
Monolingual base language file	<i>Empty</i>
Base file for new translations	po/messages.pot
File format	<i>Gettext PO file</i>

See also:

Gettext on Wikipedia, PO Files, *Update ALL_LINGUAS variable in the configure file*, *Customize gettext output*, *Update LINGUAS file*, *Generate MO files*, *Update PO files to match POT (msgmerge)*,

Monolingual Gettext

Some projects decide to use Gettext as monolingual formats - they code just IDs in their source code and the string needs to be translated to all languages, including English. Weblate does support this, though you have to choose explicitly this file format when importing components into Weblate.

The monolingual gettext PO file typically looks like:

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-monday"
msgstr "Pondělí"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-tuesday"
msgstr "Úterý"
```

(continues on next page)

star).

⁵ Location of string in source code might help skilled translators to figure out how the string is used.

⁸ See *Customizing behavior*

⁶ Additional states supported by the file format in addition to not translated and translated.

⁹ The Gettext type comments are used as flags.

¹⁰ Only max-length:N is supported as that is defined in the Xliff standard, see *Maximum Length*

⁷ XML comment placed before the <string> element is parsed as a developer comment.

(continued from previous page)

```
#: weblate/accounts/avatar.py:163
msgid "none-user"
msgstr "Žádný"
```

While the base language file will be:

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-monday"
msgstr "Monday"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-tuesday"
msgstr "Tuesday"

#: weblate/accounts/avatar.py:163
msgid "none-user"
msgstr "None"
```

Typical Weblate <i>Component configuration</i>	
File mask	po/* .po
Monolingual base language file	po/en .po
Base file for new translations	po/messages .pot
File format	<i>Gettext PO file (monolingual)</i>

4.9.5 XLIFF

XML-based format created to standardize translation files, but in the end it is one of many standards in this area.

XLIFF is usually used as bilingual, but Weblate supports it as monolingual as well.

Translations states

Changed in version 3.3: Weblate did ignore the state attribute prior to the 3.3 release.

The `state` attribute in the file is partially processed and mapped to needs edit state in Weblate (the following states are used to flag the string as needing edit if there is some target present: `new`, `needs-translation`, `needs-adaptation`, `needs-l10n`). Should the `state` attribute be missing a string is considered translated as soon as a `<target>` element exists.

Also if the translation string has `approved="yes"` it will be imported into Weblate as approved, anything else will be imported as waiting for review (which matches XLIFF specification).

That means that when using XLIFF format, it is strongly recommended to enable Weblate review process, in order to see and change the approved state of strings. See [Dedicated reviewers](#).

Similarly on importing such files, you should choose *Import as translated* under *Processing of strings needing review*.

Whitespace and newlines in XLIFF

Generally the XML formats do not differentiate between types or amounts of whitespace. If you want to keep it, you have to add the `xml:space="preserve"` flag to the string.

For example:

```
<trans-unit id="10" approved="yes">
  <source xml:space="preserve">hello</source>
  <target xml:space="preserve">Hello, world!
</target>
</trans-unit>
```

Typical Weblate <i>Component configuration</i> for bilingual XLIFF	
File mask	localizations/*.xliff
Monolingual base language file	<i>Empty</i>
Base file for new translations	localizations/en-US.xliff
File format	<i>XLIFF Translation File</i>

Typical Weblate <i>Component configuration</i> for monolingual XLIFF	
File mask	localizations/*.xliff
Monolingual base language file	localizations/en-US.xliff
Base file for new translations	localizations/en-US.xliff
File format	<i>XLIFF Translation File</i>

See also:

[XLIFF on Wikipedia](#), [XLIFF](#)

4.9.6 Java properties

Native Java format for translations.

Java properties are usually used as monolingual.

Weblate supports ISO-8859-1, UTF-8 and UTF-16 variants of this format. All of them supports storing all Unicode characters, it's just differently encoded. In the ISO-8859-1 the Unicode escape sequences are used (eg. zkou\u0161ka), all others encode characters directly either in UTF-8 or UTF-16.

Note: Loading of escape sequences will work in UTF-8 mode as well, so please be careful choosing correct charset matching your application needs.

Typical Weblate <i>Component configuration</i>	
File mask	src/app/Bundle_*.properties
Monolingual base language file	src/app/Bundle.properties
Base file for new translations	<i>Empty</i>
File format	<i>Java Properties (ISO-8859-1)</i>

See also:

[Java properties on Wikipedia](#), [Mozilla and Java properties files](#), *Formats the Java properties file*, *Cleanup translation files*,

4.9.7 Joomla translations

New in version 2.12.

Native Joomla format for translations.

Joomla translations are usually used as monolingual.

Typical Weblate <i>Component configuration</i>	
File mask	language/*/com_foobar.ini
Monolingual base language file	language/en-GB/com_foobar.ini
Base file for new translations	<i>Empty</i>
File format	<i>Joomla Language File</i>

See also:

[Specification of Joomla language files, Mozilla and Java properties files](#)

4.9.8 Qt Linguist .ts

Translation format used in Qt based applications.

Qt Linguist files are used as both bilingual and monolingual.

Typical Weblate <i>Component configuration</i> when using as bilingual	
File mask	i18n/app.*.ts
Monolingual base language file	<i>Empty</i>
Base file for new translations	i18n/app.de.ts
File format	<i>Qt Linguist Translation File</i>

Typical Weblate <i>Component configuration</i> when using as monolingual	
File mask	i18n/app.*.ts
Monolingual base language file	i18n/app.en.ts
Base file for new translations	i18n/app.en.ts
File format	<i>Qt Linguist Translation File</i>

See also:

[Qt Linguist manual](#), [Qt .ts](#), [Bilingual and monolingual formats](#)

4.9.9 Android string resources

Android specific file format for translating applications.

Android string resources are monolingual, the *Monolingual base language file* file is stored in a different location from the others `res/values/strings.xml`.

Typical Weblate <i>Component configuration</i>	
File mask	res/values-*/strings.xml
Monolingual base language file	res/values/strings.xml
Base file for new translations	<i>Empty</i>
File format	<i>Android String Resource</i>

See also:

[Android string resources documentation](#), [Android string resources](#)

Note: Android *string-array* structures are not currently supported. To work around this, you can break your string arrays apart:

```
<string-array name="several_strings">
  <item>First string</item>
  <item>Second string</item>
</string-array>
```

become:

```
<string-array name="several_strings">
  <item>@string/several_strings_0</item>
  <item>@string/several_strings_1</item>
</string-array>
<string name="several_strings_0">First string</string>
<string name="several_strings_1">Second string</string>
```

The *string-array* that points to the *string* elements should be stored in a different file, and not localized.

This script may help pre-process your existing strings.xml files and translations: <https://gist.github.com/paour/11291062>

4.9.10 Apple iOS strings

Apple specific file format for translating applications, used for both iOS and iPhone/iPad application translations.

Apple iOS strings are usually used as bilingual.

Typical Weblate <i>Component configuration</i>	
File mask	Resources/*.lproj/Localizable.strings
Monolingual base language file	Resources/en.lproj/Localizable.strings
Base file for new translations	<i>Empty</i>
File format	<i>iOS Strings (UTF-8)</i>

See also:

[Apple Strings Files documentation](#), [Mac OSX strings](#)

4.9.11 PHP strings

PHP translations are usually monolingual, so it is recommended to specify base file with English strings.

Example file:

```
<?php
$LANG['foo'] = 'bar';
$LANG['foo1'] = 'foo bar';
$LANG['foo2'] = 'foo bar baz';
$LANG['foo3'] = 'foo bar baz bag';
```

Typical Weblate <i>Component configuration</i>	
File mask	lang/*/texts.php
Monolingual base language file	lang/en/texts.php
Base file for new translations	lang/en/texts.php
File format	<i>PHP strings</i>

Note: Translate-toolkit currently has some limitations in processing PHP files, so please double check that your files won't get corrupted before using Weblate in production setup.

Following things are known to be broken:

- Adding new strings to translation, every translation has to contain all strings (even if empty).
- Handling of special chars like newlines.

See also:

PHP

4.9.12 JSON files

New in version 2.0.

Changed in version 2.16: Since Weblate 2.16 and with translate-toolkit at least 2.2.4 nested structure JSON files are supported as well.

JSON format is used mostly for translating applications implemented in JavaScript.

Weblate currently supports several variants of JSON translations:

- Simple key / value files.
- Files with nested keys.
- *JSON i18next files*
- *WebExtension JSON*

JSON translations are usually monolingual, so it is recommended to specify base file with English strings.

Example file:

```
{
  "Hello, world!\n": "Ahoj světe!\n",
  "Orangutan has %d banana.\n": "",
  "Try Weblate at https://demo.weblate.org/!\n": "",
  "Thank you for using Weblate.": ""
}
```

Nested files are supported as well (see above for requirements), such file can look like:

```
{
  "weblate": {
    "hello": "Ahoj světe!\n",
    "orangutan": "",
    "try": "",
    "thanks": ""
  }
}
```

(continues on next page)

(continued from previous page)

```
}  
}
```

Typical Weblate <i>Component configuration</i>	
File mask	langs/translation-*.json
Monolingual base language file	langs/translation-en.json
Base file for new translations	<i>Empty</i>
File format	<i>JSON nested structure file</i>

See also:

[JSON](#), [Customize JSON output](#), [Cleanup translation files](#),

4.9.13 JSON i18next files

Changed in version 2.17: Since Weblate 2.17 and with translate-toolkit at least 2.2.5 i18next JSON files with plurals are supported as well.

[i18next](#) is an internationalization-framework written in and for JavaScript. Weblate supports it's localization files with features such as plurals.

i18next translations are monolingual, so it is recommended to specify base file with English strings.

Example file:

```
{  
  "hello": "Hello",  
  "apple": "I have an apple",  
  "apple_plural": "I have {{count}} apples",  
  "apple_negative": "I have no apples"  
}
```

Typical Weblate <i>Component configuration</i>	
File mask	langs/*.json
Monolingual base language file	langs/en.json
Base file for new translations	<i>Empty</i>
File format	<i>i18next JSON file</i>

See also:

[JSON](#), [i18next JSON Format](#), [Customize JSON output](#), [Cleanup translation files](#),

4.9.14 WebExtension JSON

New in version 2.16: This is supported since Weblate 2.16 and with translate-toolkit at least 2.2.4.

File format used when translating extensions for Google Chrome or Mozilla Firefox.

Example file:

```
{
  "hello": {
    "message": "Ahoj světe!\n",
    "description": "Description"
  },
  "orangutan": {
    "message": "",
    "description": "Description"
  },
  "try": {
    "message": "",
    "description": "Description"
  },
  "thanks": {
    "message": "",
    "description": "Description"
  }
}
```

Typical Weblate <i>Component configuration</i>	
File mask	<code>_locales/*/messages.json</code>
Monolingual base language file	<code>_locales/en/messages.json</code>
Base file for new translations	<i>Empty</i>
File format	<i>WebExtension JSON file</i>

See also:

JSON, [Google chrome.i18n](#), [Mozilla Extensions Internationalization](#)

4.9.15 .Net Resource files

New in version 2.3.

.Net Resource (.resx) file is a monolingual XML file format used in Microsoft .Net Applications. It works with .resw files as well as they use identical syntax to .resx.

Typical Weblate <i>Component configuration</i>	
File mask	<code>Resources/Language.*.resx</code>
Monolingual base language file	<code>Resources/Language.resx</code>
Base file for new translations	<i>Empty</i>
File format	<i>.Net resource file</i>

See also:

.NET Resource files (.resx), [Cleanup translation files](#),

4.9.16 CSV files

New in version 2.4.

CSV files can contain a simple list of source and translation. Weblate supports the following files:

- Files with header defining fields (source, translation, location, ...). This is recommended approach as it's least error prone.

- Files with two fields - source and translation (in this order), choose *Simple CSV file* as file format
- Files with fields as defined by translate-toolkit: location, source, target, id, fuzzy, context, translator_comments, developer_comments

Warning: The CSV format currently automatically detects dialect of the CSV file. In some cases the automatic detection might fail and you will get mixed results. This is especially true for the CSV files with newlines in the values. As a workaround it is recommended to avoid omitting quoting chars.

Example file:

```
Thank you for using Weblate.,Děkujeme za použití Weblate.
```

Typical Weblate <i>Component configuration</i>	
File mask	locale/*.csv
Monolingual base language file	<i>Empty</i>
Base file for new translations	locale/en.csv
File format	<i>CSV file</i>

See also:

CSV

4.9.17 YAML files

New in version 2.9.

The plain YAML files with string keys and values.

Example YAML file:

```
weblate:
  hello: ""
  orangutan: ""
  try: ""
  thanks: ""
```

Typical Weblate <i>Component configuration</i>	
File mask	translations/messages/*.yaml
Monolingual base language file	translations/messages.en.yaml
Base file for new translations	<i>Empty</i>
File format	<i>YAML file</i>

See also:

YAML, *Ruby YAML files*

4.9.18 Ruby YAML files

New in version 2.9.

Ruby i18n YAML files with language as root node.

Example Ruby i18n YAML file:

```
cs:
  weblate:
    hello: ""
    orangutan: ""
    try: ""
    thanks: ""
```

Typical Weblate <i>Component configuration</i>	
File mask	translations/messages/*.yaml
Monolingual base language file	translations/messages.en.yaml
Base file for new translations	<i>Empty</i>
File format	<i>Ruby YAML file</i>

See also:

[YAML](#), [YAML files](#)

4.9.19 DTD files

New in version 2.18.

Example DTD file:

```
<!ENTITY hello "">
<!ENTITY orangutan "">
<!ENTITY try "">
<!ENTITY thanks "">
```

Typical Weblate <i>Component configuration</i>	
File mask	locale/*.dtd
Monolingual base language file	locale/en.dtd
Base file for new translations	<i>Empty</i>
File format	<i>DTD file</i>

See also:

[Mozilla DTD format](#)

4.9.20 Windows RC files

New in version 3.0: Experimental support has been added in Weblate 3.0, not supported on Python 3.

Example Windows RC file:

```
LANGUAGE LANG_CZECH, SUBLANG_DEFAULT

STRINGTABLE DISCARDABLE
BEGIN

IDS_MSG1 "Hello, world!\n"
IDS_MSG2 "Orangutan has %d banana.\n"
```

(continues on next page)

(continued from previous page)

```
IDS_MSG3 "Try Weblate at http://demo.weblate.org/!\n"
IDS_MSG4 "Thank you for using Weblate."
END
```

Typical Weblate <i>Component configuration</i>	
File mask	lang/*.rc
Monolingual base language file	lang/en-US.rc
Base file for new translations	lang/en-US.rc
File format	<i>RC file</i>

See also:[Windows RC files](#)

4.9.21 App store metadata files

New in version 3.5.

Weblate can translate metadata used for publishing apps in various app stores. Currently it is known to be compatible with following tools:

- [Triple-T gradle-play-publisher](#)
- [Fastlane](#)
- [F-Droid](#)

The metadata consist of several text files which Weblate will present as separate strings to translate.

Typical Weblate <i>Component configuration</i>	
File mask	fastlane/android/metadata/*
Monolingual base language file	fastlane/android/metadata/en-US
Base file for new translations	fastlane/android/metadata/en-US
File format	<i>App store metadata files</i>

4.9.22 Subtitle files

New in version 3.7.

Weblate can translate various subtitle files:

- SubRip subtitle file (*.srt)
- MicroDVD subtitles file (*.sub)
- Advanced Substation Alpha subtitles file (*.ass)
- Substation Alpha subtitles file (*.ssa)

Typical Weblate <i>Component configuration</i>	
File mask	path/*.srt
Monolingual base language file	path/en.srt
Base file for new translations	path/en.srt
File format	<i>SubRip subtitle file</i>

See also:

[Subtitles](#)

4.9.23 Excel Open XML

New in version 3.2.

Weblate can import and export Excel Open XML (xlsx) files.

When using xlsx files for translation upload, be aware that only the active worksheet is considered and there must be at least a column called `source` (which contains the source string) and a column called `target` (which contains the translation). Additionally there should be the column `context` (which contains the context path of the translation string). If you use the xlsx download for exporting the translations into an Excel workbook, you already get a file with the correct file format.

4.9.24 Others

Most formats supported by translate-toolkit which support serializing can be easily supported, but they did not (yet) receive any testing. In most cases some thin layer is needed in Weblate to hide differences in behavior of different translate-toolkit storages.

See also:

[Translation Related File Formats](#)

4.9.25 Adding new translations

Changed in version 2.18: In versions prior to 2.18 the behaviour of adding new translations was file format specific.

Weblate can automatically start new translation for all of the file formats.

Some formats expect to start with empty file and only translated strings to be included (eg. *Android string resources*), while others expect to have all keys present (eg. *GNU Gettext*). In some situations this really doesn't depend on the format, but rather on framework you use to handle the translation (eg. with *JSON files*).

When you specify *Base file for new translations* in *Component configuration*, Weblate will use this file to start new translations. Any exiting translations will be removed from the file when doing so.

When *Base file for new translations* is empty and file format supports it, empty file is created where new strings will be added once they are translated.

The *Language code style* allows you to customize language code used in generated filenames:

Default based on the file format Dependent on file format, for most of them POSIX is used.

POSIX style using underscore as a separator Typically used by Gettext and related tools, produces language codes like *pt-BR*.

BCP style using hyphen as a separator Typically used on web platforms, produces language codes like *pt_BR*.

Android style Used only on Android apps, produces language codes like *pt-rBR*.

4.10 Version control integration

Weblate currently supports *Git* (with extended support for *GitHub*, *Gerrit* and *Subversion*) and *Mercurial* as version control backends.

4.10.1 Accessing repositories

The VCS repository you want to use has to be accessible to Weblate. With a publicly available repository you just need to enter correct URL (for example `git://github.com/WeblateOrg/weblate.git` or `https://github.com/WeblateOrg/weblate.git`), but for private repositories the setup might be more complex.

Weblate internal URLs

To share one repository between different components you can use a special URL like `weblate://project/component`. This way, the component will share the VCS repository configuration with referenced component and the VCS repository will be stored just once on the disk.

SSH repositories

The most frequently used method to access private repositories is based on SSH. To have access to such a repository, you generate SSH key for Weblate and authorize it to access the repository. Weblate also needs to know the host key to avoid man in the middle attacks. This all can be done in the Weblate administration interface:

The screenshot shows the 'Weblate administration' interface. At the top, there's a navigation bar with links: 'WELCOME, WEBLATE TEST', 'VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT'. Below this, a breadcrumb trail shows 'Home > SSH keys'. The main heading is 'SSH keys management' with a 'DOCUMENTATION' button. Under 'Public SSH key', it states 'Weblate currently uses following SSH key:' and displays a long SSH key string. Below that, under 'Known host keys', there's a table with columns 'HOSTNAME', 'KEY TYPE', and 'FINGERPRINT'. The table contains one entry for 'github.com' with key type 'ssh-rsa' and a fingerprint. At the bottom, there's an 'Add host key' section with a text box for 'Host:' and a 'Port:' field, followed by a 'Submit' button.

Generating SSH keys

You can generate or display the key currently used by Weblate in the admin interface (follow *SSH keys* link on main admin page). Once you've done this, Weblate should be able to access your repository.

Note: The keys need to be without password to make it work, so be sure they are well protected against malicious usage.

Warning: On GitHub, you can add the key to only one repository. See the following sections for other solutions for GitHub.

Verifying SSH host keys

Before connecting to the repository, you also need to verify SSH host keys of servers you are going to access in the same section of the admin interface. You can do this in the *Add host key* section. Just enter hostname you are going to access (eg. `gitlab.com`) and press *Submit*. After adding it please verify that the fingerprint matches the server you're adding, the fingerprints will be displayed in the confirmation message:

Weblate administration
WELCOME, WEBLATE TEST. VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT

Home > SSH keys

Added host key for github.com with fingerprint nThbg6kXUpJWGI7E1IGOCspRomTxdCARLviKw6E5SY8 (ssh-rsa), please verify that it is correct.

SSH keys management
DOCUMENTATION

Public SSH key

Weblate currently uses following SSH key:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQADn3tsDWDdyxK8NsQPTNYID/tuXBHMBjvsjo2rNLQFwRj6cStZH53GJQ+yCffc5v5TzV0E8MAZ8zJG8wLknhy+1HX4cNjbA+fxDEaCXWPbX9uJ0X14AK836b7Jccp
9nn3xjNVgOgipDQdQUycy5uBFeX3xi2r97zkeABRlm1bJOBclXMxkyTp/VQG7OpBPIEjYIU3jbMvwPpR0vM5FZ8/lrIO7lm1jL22bVpbCKrc+fu4eyH
```

Known host keys

HOSTNAME	KEY TYPE	FINGERPRINT
github.com	ssh-rsa	nThbg6kXUpJWGI7E1IGOCspRomTxdCARLviKw6E5SY8

Add host key

To access SSH hosts, its host key needs to be verified. You can get the host key by entering a domain name or IP for the host in the form below.

Host: Port:

Submit

HTTPS repositories

To access protected HTTPS repositories, you need to include the username and password in the URL. Don't worry, Weblate will strip this information when showing the URL to the users (if they are allowed to see the repository URL at all).

For example the GitHub URL with authentication might look like `https://user:your_access_token@github.com/WeblateOrg/weblate.git`.

Note: In case your username or password contains special chars, those have to be URL encoded, for example `https://user%40example.com:%24password%23@bitbucket.org/...``.

Using proxy

If you need to access http/https VCS repositories using a proxy server, you need to configure the VCS to use it.

This can be configured using the `http_proxy`, `https_proxy`, and `all_proxy` environment variables (check cURL documentation for more details) or by enforcing it in VCS configuration, for example:

```
git config --global http.proxy http://user:password@proxy.example.com:80
```

Note: The proxy setting needs to be done in the same context which is used to execute Weblate. For the environment it should be set for both wsgi and Celery servers. The VCS configuration has to be set for the user which is running

Weblate.

See also:

[curl manpage](#), [git config documentation](#)

4.10.2 Git

Git is first VCS backend that was available in Weblate and is still the most stable and tested one.

See also:

See [Accessing repositories](#) for information how to access different kind of repositories.

GitHub repositories

You can access GitHub repositories by SSH as mentioned above, but in case you need to access more repositories, you will hit a GitHub limitation on the SSH key usage (one key can be used only for one repository). There are several ways to work around this limitation.

For smaller deployments, you can use HTTPS authentication using a personal access token and your account, see [Creating an access token for command-line use](#).

For a bigger setup, it is usually better to create dedicated user for Weblate, assign him the SSH key generated in Weblate and grant him access to all repositories you want.

Customizing Git configuration

Weblate invokes all VCS commands with HOME pointed to home directory in `DATA_DIR`, therefore if you want to edit user configuration, you need to do this in `DATA_DIR/home/.git`.

Git remote helpers

You can also use Git [remote helpers](#) for supporting other VCS as well, but this usually leads to other problems, so be prepared to debug them.

At this time, helpers for Bazaar and Mercurial are available within separate repositories on GitHub: [git-remote-hg](#) and [git-remote-bzr](#). You can download them manually and put somewhere in your search path (for example `~/bin`). You also need to have installed appropriate version control programs as well.

Once you have these installed, you can use such remotes to specify repository in Weblate.

To clone `gnuhello` project from Launchpad with Bazaar use:

```
bzr::lp:gnuhello
```

For `hello` repository from `selenic.com` with Mercurial use:

```
hg::http://selenic.com/repo/hello
```

Warning: Please be prepared to some inconvenience when using Git remote helpers, for example with Mercurial, the remote helper sometimes tends to create new tip when pushing changes back.

4.10.3 GitHub

New in version 2.3.

This just adds a thin layer on top of *Git* to allow push translation changes as pull requests instead of pushing directory to the repository. It currently uses the *hub* tool to do the integration.

There is no need to use this to access Git repositories, ordinary *Git* works the same, the only difference is how pushing to a repository is handled. With *Git* changes are pushed directly to the repository, while *GitHub* creates pull requests.

Pushing changes to GitHub as pull request

If you are translating a project that's hosted on GitHub and don't want to push translations to the repository, you can have them sent as a pull request instead.

You need to configure the *hub* command line tool and set *GITHUB_USERNAME* for this to work.

See also:

GITHUB_USERNAME, *Setting up hub* for configuration instructions

Setting up hub

Pushing changes to GitHub as pull request requires a configured *hub* installation on your server. Follow the installation instructions at <https://hub.github.com/> and perform an action with *hub* to finish the configuration, for example:

```
# DATA_DIR is set in Weblate settings.py, set it accordingly.
# Is is /app/data in Docker
HOME=${DATA_DIR}/home hub clone octocat/Spoon-Knife
```

The *hub* will ask you for your GitHub credentials, retrieve a token and store it into `~/.config/hub`. This file has to be readable by user running Weblate.

Note: Use the username you configured *hub* with as *GITHUB_USERNAME* (*WEBLATE_GITHUB_USERNAME* for the Docker image).

4.10.4 Gerrit

New in version 2.2.

Adds a thin layer atop *Git* to allow pushing translation changes as Gerrit review requests, instead of pushing a directory to the repository. Currently uses the *git-review* tool to do the integration.

Please refer to the Gerrit documentation for setting up the repository with necessary configuration.

4.10.5 Mercurial

New in version 2.1.

Mercurial is another VCS you can use directly in Weblate.

Note: It should work with any Mercurial version, but there are sometimes incompatible changes to the command line interface which break Weblate.

See also:

See [Accessing repositories](#) for information how to access different kind of repositories.

4.10.6 Subversion

New in version 2.8.

Thanks to [git-svn](#), Weblate can work with [subversion](#) repositories. Git-svn is a Perl script that enables the usage of subversion with a git client, enabling users to have a full clone of the internal repository and commit locally.

Note: Weblate tries to detect Subversion repository layout automatically - it supports both direct URLs for branch or repositories with standard layout (branches/, tags/ and trunk/). See [git-svn documentation](#) for more information.

Changed in version 2.19: In older versions only repositories with standard layout were supported.

Subversion Credentials

Weblate expects you to have accepted the certificate upfront and inserted your credential, if needed. It will look into the DATA_DIR directory. To insert your credential and accept the certificate, you can run svn once with the *\$HOME* environment variable set to the DATA_DIR:

```
HOME=${DATA_DIR}/home svn co https://svn.example.com/example
```

See also:

[DATA_DIR](#)

4.11 Weblate's Web API

4.11.1 REST API

New in version 2.6: The API is available since Weblate 2.6.

The API is accessible on the `/api/` URL and it is based on [Django REST framework](#). You can use it directly or by [Weblate Client](#).

Authentication and generic parameters

The public project API is available without authentication, though unauthenticated requests are heavily throttled (by default to 100 requests per day), so it is recommended to use authentication. The authentication uses a token, which you can get in your profile. Use it in the `Authorization` header:

ANY /

Generic request behaviour for the API, the headers, status codes and parameters here apply to all endpoints as well.

Query Parameters

- **format** – Response format (overrides [Accept](#)). Possible values depends on REST framework setup, by default `json` and `api` are supported. The latter provides web browser interface for API.

Request Headers

- [Accept](#) – the response content type depends on [Accept](#) header
- [Authorization](#) – optional token to authenticate

Response Headers

- [Content-Type](#) – this depends on [Accept](#) header of request
- [Allow](#) – list of allowed HTTP methods on object

Response JSON Object

- **detail** (*string*) – verbose description of failure (for HTTP status codes other than [200 OK](#))
- **count** (*int*) – total item count for object lists
- **next** (*string*) – next page URL for object lists
- **previous** (*string*) – previous page URL for object lists
- **results** (*array*) – results for object lists
- **url** (*string*) – URL to access this resource using API
- **web_url** (*string*) – URL to access this resource using web browser

Status Codes

- [200 OK](#) – when request was correctly handled
- [400 Bad Request](#) – when form parameters are missing
- [403 Forbidden](#) – when access is denied
- [429 Too Many Requests](#) – when throttling is in place

Authentication examples

Example request:

```
GET /api/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Authorization: Token YOUR-TOKEN
```

Example response:

```
HTTP/1.0 200 OK
Date: Fri, 25 Mar 2016 09:46:12 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
```

(continues on next page)

(continued from previous page)

```
Allow: GET, HEAD, OPTIONS
```

```
{
  "projects": "http://example.com/api/projects/",
  "components": "http://example.com/api/components/",
  "translations": "http://example.com/api/translations/",
  "languages": "http://example.com/api/languages/"
}
```

CURL example:

```
curl \
  -H "Authorization: Token TOKEN" \
  https://example.com/api/
```

Passing Parameters Examples

For the **POST** method the parameters can be specified either as form submission (*application/x-www-form-urlencoded*) or as JSON (*application/json*).

Form request example:

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Token TOKEN

operation=pull
```

JSON request example:

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"operation": "pull"}
```

CURL example:

```
curl \
  -d operation=pull \
  -H "Authorization: Token TOKEN" \
  http://example.com/api/components/hello/weblate/repository/
```

CURL JSON example:

```
curl \
  --data-binary '{"operation": "pull"}' \
  -H "Content-Type: application/json" \
  -H "Authorization: Token TOKEN" \
  http://example.com/api/components/hello/weblate/repository/
```

Rate limiting

The API requests are rate limited; the default configuration limits it to 100 requests per day for anonymous users and 1000 requests per day for authenticated users.

Rate limiting can be adjusted in the `settings.py`; see [Throttling in Django REST framework documentation](#) for more details how to configure it.

API Entry Point

GET `/api/`

The API root entry point.

Example request:

```
GET /api/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Authorization: Token YOUR-TOKEN
```

Example response:

```
HTTP/1.0 200 OK
Date: Fri, 25 Mar 2016 09:46:12 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, HEAD, OPTIONS

{
  "projects": "http://example.com/api/projects/",
  "components": "http://example.com/api/components/",
  "translations": "http://example.com/api/translations/",
  "languages": "http://example.com/api/languages/"
}
```

Languages

GET `/api/languages/`

Returns a list of all languages.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Language object attributes are documented at `GET /api/languages/(string:language)/`.

GET `/api/languages/(string: language)/`

Returns information about a language.

Parameters

- **language** (*string*) – Language code

Response JSON Object

- **code** (*string*) – Language code
- **direction** (*string*) – Text direction

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Example JSON data:

```
{
  "code": "en",
  "direction": "ltr",
  "name": "English",
  "url": "http://example.com/api/languages/en/",
  "web_url": "http://example.com/languages/en/"
}
```

Projects

GET /api/projects/

Returns a list of all projects.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Project object attributes are documented at [GET /api/projects/\(string:project\)/](#).

GET /api/projects/(string: project) /

Returns information about a project.

Parameters

- **project** (*string*) – Project URL slug

Response JSON Object

- **name** (*string*) – project name
- **slug** (*string*) – project slug
- **source_language** (*object*) – source language object; see [GET /api/languages/\(string:language\)/](#)
- **web** (*string*) – project website
- **components_list_url** (*string*) – URL to components list; see [GET /api/projects/\(string:project\)/components/](#)
- **repository_url** (*string*) – URL to repository status; see [GET /api/projects/\(string:project\)/repository/](#)
- **changes_list_url** (*string*) – URL to changes list; see [GET /api/projects/\(string:project\)/changes/](#)

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Example JSON data:

```
{
  "name": "Hello",
  "slug": "hello",
  "source_language": {
    "code": "en",
    "direction": "ltr",
    "name": "English",
    "url": "http://example.com/api/languages/en/",
    "web_url": "http://example.com/languages/en/"
  },
  "url": "http://example.com/api/projects/hello/",
  "web": "https://weblate.org/",
  "web_url": "http://example.com/projects/hello/"
}
```

GET `/api/projects/(string: project)/changes/`
Returns a list of project changes.

Parameters

- **project** (*string*) – Project URL slug

Response JSON Object

- **results** (*array*) – array of component objects; see `GET /api/changes/(int:pk)/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/projects/(string: project)/repository/`

Returns information about VCS repository status. This endpoint contains only an overall summary for all repositories for the project. To get more detailed status use `GET /api/components/(string:project)/(string:component)/repository/`.

Parameters

- **project** (*string*) – Project URL slug

Response JSON Object

- **needs_commit** (*boolean*) – whether there are any pending changes to commit
- **needs_merge** (*boolean*) – whether there are any upstream changes to merge
- **needs_push** (*boolean*) – whether there are any local changes to push

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Example JSON data:

```
{
  "needs_commit": true,
  "needs_merge": false,
  "needs_push": true
}
```

POST `/api/projects/(string: project)/repository/`

Performs given operation on the VCS repository.

Parameters

- **project** (*string*) – Project URL slug

Request JSON Object

- **operation** (*string*) – Operation to perform: one of push, pull, commit, reset, cleanup

Response JSON Object

- **result** (*boolean*) – result of the operation

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

CURL example:

```
curl \
  -d operation=pull \
  -H "Authorization: Token TOKEN" \
  http://example.com/api/components/hello/weblate/repository/
```

JSON request example:

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"operation": "pull"}
```

JSON response example:

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{"result": true}
```

GET `/api/projects/(string: project)/components/`

Returns a list of translation components in the given project.

Parameters

- **project** (*string*) – Project URL slug

Response JSON Object

- **results** (*array*) – array of component objects; see [GET /api/components/\(string:project\)/\(string:component\)/](#)

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project)/statistics/`
Returns paginated statistics for all languages within a project.

New in version 2.10.

Parameters

- **project** (*string*) – Project URL slug

Response JSON Object

- **results** (*array*) – array of translation statistics objects
- **language** (*string*) – language name
- **code** (*string*) – language code
- **total** (*int*) – total number of strings
- **translated** (*int*) – number of translated strings
- **translated_percent** (*float*) – percentage of translated strings
- **total_words** (*int*) – total number of words
- **translated_words** (*int*) – number of translated words
- **words_percent** (*float*) – percentage of translated words

Components

GET `/api/components/`
Returns a list of translation components.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Component object attributes are documented at `GET /api/components/(string:project)/(string:component)/`.

GET `/api/components/(string: project) /`
string: *component* / Returns information about translation component.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **branch** (*string*) – VCS repository branch
- **file_format** (*string*) – file format of translations
- **filemask** (*string*) – mask of translation files in the repository
- **git_export** (*string*) – URL of the exported VCS repository with translations
- **license** (*string*) – license for translations

- **license_url** (*string*) – URL of license for translations
- **name** (*string*) – name of component
- **slug** (*string*) – slug of component
- **project** (*object*) – the translation project; see `GET /api/projects/(string:project)/`
- **repo** (*string*) – VCS repository URL
- **template** (*string*) – base file for monolingual translations
- **new_base** (*string*) – base file for adding new translations
- **vcs** (*string*) – version control system
- **repository_url** (*string*) – URL to repository status; see `GET /api/components/(string:project)/(string:component)/repository/`
- **translations_url** (*string*) – URL to translations list; see `GET /api/components/(string:project)/(string:component)/translations/`
- **lock_url** (*string*) – URL to lock status; see `GET /api/components/(string:project)/(string:component)/lock/`
- **changes_list_url** (*string*) – URL to changes list; see `GET /api/components/(string:project)/(string:component)/changes/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Example JSON data:

```
{
  "branch": "master",
  "file_format": "po",
  "filemask": "po/*.po",
  "git_export": "",
  "license": "",
  "license_url": "",
  "name": "Weblate",
  "slug": "weblate",
  "project": {
    "name": "Hello",
    "slug": "hello",
    "source_language": {
      "code": "en",
      "direction": "ltr",
      "name": "English",
      "url": "http://example.com/api/languages/en/",
      "web_url": "http://example.com/languages/en/"
    },
    "url": "http://example.com/api/projects/hello/",
    "web": "https://weblate.org/",
    "web_url": "http://example.com/projects/hello/"
  },
  "repo": "file:///home/nijel/work/weblate-hello",
  "template": "",
  "new_base": "",
  "url": "http://example.com/api/components/hello/weblate/",
```

(continues on next page)

(continued from previous page)

```

    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
  }

```

GET `/api/components/(string: project) /`
string: `component/changes/` Returns a list of component changes.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **results** (*array*) – array of component objects; see `GET /api/changes/(int:pk)/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/lock/` Returns component lock status.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **locked** (*boolean*) – whether component is locked for updates

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Example JSON data:

```

{
  "locked": false
}

```

POST `/api/components/(string: project) /`
string: `component/lock/` Sets component lock status.

Response is same as `GET /api/components/(string:project)/(string:component)/lock/`.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Request JSON Object

- **lock** – Boolean whether to lock or not.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/repository/` Returns information about VCS repository status.

The response is same as for `GET /api/projects/(string:project)/repository/`.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **needs_commit** (*boolean*) – whether there are any pending changes to commit
- **needs_merge** (*boolean*) – whether there are any upstream changes to merge
- **needs_push** (*boolean*) – whether there are any local changes to push
- **remote_commit** (*string*) – Remote commit information
- **status** (*string*) – VCS repository status as reported by VCS
- **merge_failure** – Text describing merge failure or null if there is none

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

POST `/api/components/(string: project) /`
string: `component/repository/` Performs the given operation on a VCS repository.

See `POST /api/projects/(string:project)/repository/` for documentation.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Request JSON Object

- **operation** (*string*) – Operation to perform: one of push, pull, commit, reset, cleanup

Response JSON Object

- **result** (*boolean*) – result of the operation

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/monolingual_base/` Downloads base file for monolingual translations.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/new_template/` Downloads template file for new translations.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/translations/` Returns a list of translation objects in the given component.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **results** (*array*) – array of translation objects; see `GET /api/translations/(string:project)/(string:component)/(string:language)/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/components/(string: project) /`
string: `component/statistics/` Returns paginated statistics for all translations within component.

New in version 2.7.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug

Response JSON Object

- **results** (*array*) – array of translation statistics objects; see `GET /api/translations/(string:project)/(string:component)/(string:language)/statistics/`

Translations

GET `/api/translations/`
 Returns a list of translations.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Translation object attributes are documented at `GET /api/translations/(string:project)/(string:component)/(string:language)/`.

GET /api/translations/(string: project) /
string: component/string: language/ Returns information about a translation.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Response JSON Object

- **component** (*object*) – component object; see `GET /api/components/(string:project)/(string:component)/`
- **failing_checks** (*int*) – number of strings failing check
- **failing_checks_percent** (*float*) – percentage of strings failing check
- **failing_checks_words** (*int*) – number of words with failing check
- **filename** (*string*) – translation filename
- **fuzzy** (*int*) – number of strings marked for review
- **fuzzy_percent** (*float*) – percentage of strings marked for review
- **fuzzy_words** (*int*) – number of words marked for review
- **have_comment** (*int*) – number of strings with comment
- **have_suggestion** (*int*) – number of strings with suggestion
- **is_template** (*boolean*) – whether translation is monolingual base
- **language** (*object*) – source language object; see `GET /api/languages/(string:language)/`
- **language_code** (*string*) – language code used in the repository; this can be different from language code in the language object
- **last_author** (*string*) – name of last author
- **last_change** (*timestamp*) – last change timestamp
- **revision** (*string*) – hash revision of the file
- **share_url** (*string*) – URL for sharing leading to engage page
- **total** (*int*) – total number of strings
- **total_words** (*int*) – total number of words
- **translate_url** (*string*) – URL for translating
- **translated** (*int*) – number of translated strings
- **translated_percent** (*float*) – percentage of translated strings
- **translated_words** (*int*) – number of translated words
- **repository_url** (*string*) – URL to repository status; see `GET /api/translations/(string:project)/(string:component)/(string:language)/repository/`

- **file_url** (*string*) – URL to file object; see *GET /api/translations/(string:project)/(string:component)/(string:language)/file/*
- **changes_list_url** (*string*) – URL to changes list; see *GET /api/translations/(string:project)/(string:component)/(string:language)/changes/*
- **units_list_url** (*string*) – URL to strings list; see *GET /api/translations/(string:project)/(string:component)/(string:language)/units/*

See also:

Additional common headers, parameters and status codes are documented at *Authentication and generic parameters*.

Example JSON data:

```
{
  "component": {
    "branch": "master",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
    "license": "",
    "license_url": "",
    "name": "Weblate",
    "new_base": "",
    "project": {
      "name": "Hello",
      "slug": "hello",
      "source_language": {
        "code": "en",
        "direction": "ltr",
        "name": "English",
        "url": "http://example.com/api/languages/en/",
        "web_url": "http://example.com/languages/en/"
      },
      "url": "http://example.com/api/projects/hello/",
      "web": "https://weblate.org/",
      "web_url": "http://example.com/projects/hello/"
    },
    "repo": "file:///home/nijel/work/weblate-hello",
    "slug": "weblate",
    "template": "",
    "url": "http://example.com/api/components/hello/weblate/",
    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
  },
  "failing_checks": 3,
  "failing_checks_percent": 75.0,
  "failing_checks_words": 11,
  "filename": "po/cs.po",
  "fuzzy": 0,
  "fuzzy_percent": 0.0,
  "fuzzy_words": 0,
  "have_comment": 0,
  "have_suggestion": 0,
  "is_template": false,
  "language": {
```

(continues on next page)

(continued from previous page)

```
    "code": "cs",
    "direction": "ltr",
    "name": "Czech",
    "url": "http://example.com/api/languages/cs/",
    "web_url": "http://example.com/languages/cs/"
  },
  "language_code": "cs",
  "last_author": "Weblate Admin",
  "last_change": "2016-03-07T10:20:05.499",
  "revision": "7ddfafe6daaf57fc8654cc852ea6be212b015792",
  "share_url": "http://example.com/engage/hello/cs/",
  "total": 4,
  "total_words": 15,
  "translate_url": "http://example.com/translate/hello/weblate/cs/",
  "translated": 4,
  "translated_percent": 100.0,
  "translated_words": 15,
  "url": "http://example.com/api/translations/hello/weblate/cs/",
  "web_url": "http://example.com/projects/hello/weblate/cs/"
}
```

GET `/api/translations/(string: project) /`
string: `component/string: language/changes/` Returns a list of translation changes.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Response JSON Object

- **results** (*array*) – array of component objects; see `GET /api/changes/(int:pk)/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/translations/(string: project) /`
string: `component/string: language/units/` Returns a list of translation units.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Response JSON Object

- **results** (*array*) – array of component objects; see `GET /api/units/(int:pk)/`

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/translations/(string: project) /`
string: `component/string: language/file/` Download current translation file as stored in VCS (without `format` parameter) or as converted to a standard format (currently supported: Gettext PO, MO, XLIFF and TBX).

Note: This API endpoint uses different logic for output than rest of API as it operates on whole file rather than on data. Set of accepted `format` parameter differs and without such parameter you get translation file as stored in VCS.

Query Parameters

- **format** – File format to use; if not specified no format conversion happens; supported file formats: `po`, `mo`, `xliff`, `xliff11`, `tbx`

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

POST `/api/translations/(string: project) /`
string: `component/string: language/file/` Upload new file with translations.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Form Parameters

- **boolean overwrite** – Whether to overwrite existing translations (defaults to no)
- **file file** – Uploaded file
- **string email** – Author email
- **string author** – Author name
- **string method** – Upload method (`translate`, `approve`, `suggest`, `fuzzy`)
- **string fuzzy** – Fuzzy strings processing (`empty`, `process`, `approve`)

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

CURL example:

```
curl -X POST \
  -F file=@strings.xml \
  -H "Authorization: Token TOKEN" \
  http://example.com/api/translations/hello/android/cs/file/
```

GET `/api/translations/(string: project) /string: component/string: language/repository/` Returns information about VCS repository status.

The response is same as for `GET /api/components/(string:project)/(string:component)/repository/`.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

POST `/api/translations/(string: project) /string: component/string: language/repository/` Performs given operation on the VCS repository.

See `POST /api/projects/(string:project)/repository/` for documentation.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Request JSON Object

- **operation** (*string*) – Operation to perform: one of push, pull, commit, reset, cleanup

Response JSON Object

- **result** (*boolean*) – result of the operation

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

GET `/api/translations/(string: project) /string: component/string: language/statistics/` Returns detailed translation statistics.

New in version 2.7.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component URL slug
- **language** (*string*) – Translation language code

Response JSON Object

- **code** (*string*) – language code
- **failing** (*int*) – number of failing checks
- **failing_percent** (*float*) – percentage of failing checks

- **fuzzy** (*int*) – number of strings needing review
- **fuzzy_percent** (*float*) – percentage of strings needing review
- **total_words** (*int*) – total number of words
- **translated_words** (*int*) – number of translated words
- **last_author** (*string*) – name of last author
- **last_change** (*timestamp*) – date of last change
- **name** (*string*) – language name
- **total** (*int*) – total number of strings
- **translated** (*int*) – number of translated strings
- **translated_percent** (*float*) – percentage of translated strings
- **url** (*string*) – URL to access the translation (engagement URL)
- **url_translate** (*string*) – URL to access the translation (real translation URL)

Units

New in version 2.10.

GET /api/units/

Returns list of translation units.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Unit object attributes are documented at [GET /api/units/\(int:pk\)/](#).

GET /api/units/(int: pk) /

Returns information about translation unit.

Parameters

- **pk** (*int*) – Unit ID

Response JSON Object

- **translation** (*string*) – URL of a related translation object
- **source** (*string*) – source string
- **previous_source** (*string*) – previous source string used for fuzzy matching
- **target** (*string*) – target string
- **id_hash** (*string*) – unique identifier of the unit
- **content_hash** (*string*) – unique identifier of the source string
- **location** (*string*) – location of the unit in source code
- **context** (*string*) – translation unit context
- **comment** (*string*) – translation unit comment
- **flags** (*string*) – translation unit flags
- **fuzzy** (*boolean*) – whether unit is fuzzy or marked for review

- **translated** (*boolean*) – whether unit is translated
- **position** (*int*) – unit position in translation file
- **has_suggestion** (*boolean*) – whether unit has suggestions
- **has_comment** (*boolean*) – whether unit has comments
- **has_failing_check** (*boolean*) – whether unit has failing checks
- **num_words** (*int*) – number of source words
- **priority** (*int*) – translation priority; 100 is default
- **id** (*int*) – unit identifier
- **web_url** (*string*) – URL where unit can be edited
- **source_info** (*string*) – Source string information link; see [GET /api/sources/\(int:pk\)/](#)

Changes

New in version 2.10.

GET /api/changes/

Returns a list of translation changes.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Change object attributes are documented at [GET /api/changes/\(int:pk\)/](#).

GET /api/changes/(int: pk) /

Returns information about translation change.

Parameters

- **pk** (*int*) – Change ID

Response JSON Object

- **unit** (*string*) – URL of a related unit object
- **translation** (*string*) – URL of a related translation object
- **component** (*string*) – URL of a related component object
- **dictionary** (*string*) – URL of a related dictionary object
- **user** (*string*) – URL of a related user object
- **author** (*string*) – URL of a related author object
- **timestamp** (*timestamp*) – event timestamp
- **action** (*int*) – numeric identification of action
- **action_name** (*string*) – text description of action
- **target** (*string*) – event changed text or detail
- **id** (*int*) – change identifier

Sources

New in version 2.14.

GET `/api/sources/`

Returns a list of source string information.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Sources object attributes are documented at [GET /api/sources/\(int:pk\)/](#).

GET `/api/sources/(int: pk) /`

Returns information about source information.

Parameters

- **pk** (*int*) – Source information ID

Response JSON Object

- **id_hash** (*string*) – unique identifier of the unit
- **component** (*string*) – URL of a related component object
- **timestamp** (*timestamp*) – timestamp when source string was first seen by Weblate
- **priority** (*int*) – source string priority, 100 is default
- **check_flags** (*string*) – source string flags
- **units** (*array*) – links to units; see [GET /api/units/\(int:pk\)/](#)
- **screenshots** (*array*) – links to assigned screenshots; see [GET /api/screenshots/\(int:pk\)/](#)

Screenshots

New in version 2.14.

GET `/api/screenshots/`

Returns a list of screenshot string information.

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

Sources object attributes are documented at [GET /api/screenshots/\(int:pk\)/](#).

GET `/api/screenshots/(int: pk) /`

Returns information about screenshot information.

Parameters

- **pk** (*int*) – Screenshot ID

Response JSON Object

- **name** (*string*) – name of a screenshot
- **component** (*string*) – URL of a related component object

- **file_url** (*string*) – URL to download a file; see `GET /api/screenshots/(int:pk)/file/`
- **sources** (*array*) – link to associated source string information; see `GET /api/sources/(int:pk)/`

GET `/api/screenshots/(int: pk)/file/`

Download the screenshot image.

Parameters

- **pk** (*int*) – Screenshot ID

POST `/api/screenshots/(int: pk)/file/`

Replace screenshot image.

Parameters

- **pk** (*int*) – Screenshot ID

Form Parameters

- **file image** – Uploaded file

See also:

Additional common headers, parameters and status codes are documented at [Authentication and generic parameters](#).

CURL example:

```
curl -X POST \
  -F image=@image.png \
  -H "Authorization: Token TOKEN" \
  http://example.com/api/screenshots/1/file/
```

4.11.2 Notification hooks

Notification hooks allow external applications to notify Weblate that the VCS repository has been updated.

You can use repository endpoints for projects, components and translations to update individual repositories; see `POST /api/projects/(string:project)/repository/` for documentation.

GET `/hooks/update/(string: project) /`

string: *component/* Deprecated since version 2.6: Please use `POST /api/components/(string:project)/(string:component)/repository/` instead which works properly with authentication for ACL limited projects.

Triggers update of a component (pulling from VCS and scanning for translation changes).

GET `/hooks/update/(string: project) /`

Deprecated since version 2.6: Please use `POST /api/projects/(string:project)/repository/` instead which works properly with authentication for ACL limited projects.

Triggers update of all components in a project (pulling from VCS and scanning for translation changes).

POST `/hooks/github/`

Special hook for handling GitHub notifications and automatically updating matching components.

Note: GitHub includes direct support for notifying Weblate: enable Weblate service hook in repository settings and set the URL to the URL of your Weblate installation.

See also:

Automatically receiving changes from GitHub For instruction on setting up GitHub integration

<https://help.github.com/articles/creating-webhooks> Generic information about GitHub Webhooks

ENABLE_HOOKS For enabling hooks for whole Weblate

POST `/hooks/gitlab/`

Special hook for handling GitLab notifications and automatically updating matching components.

See also:

Automatically receiving changes from GitLab For instruction on setting up GitLab integration

<https://docs.gitlab.com/ce/user/project/integrations/webhooks.html> Generic information about GitLab Webhooks

ENABLE_HOOKS For enabling hooks for whole Weblate

POST `/hooks/bitbucket/`

Special hook for handling Bitbucket notifications and automatically updating matching components.

See also:

Automatically receiving changes from Bitbucket For instruction on setting up Bitbucket integration

<https://confluence.atlassian.com/bitbucket/manage-webhooks-735643732.html> Generic information about Bitbucket Webhooks

ENABLE_HOOKS For enabling hooks for whole Weblate

POST `/hooks/pagure/`

Special hook for handling Pagure notifications and automatically updating matching components.

See also:

Automatically receiving changes from Pagure For instruction on setting up Bitbucket integration

https://docs.pagure.org/pagure/usage/using_webhooks.html Generic information about Bitbucket Webhooks

ENABLE_HOOKS For enabling hooks for whole Weblate

4.11.3 Exports

Weblate provides various exports to allow you to further process the data.

GET `/exports/stats/ (string: project) /`
string: `component/`

Query Parameters

- **format** (*string*) – Output format: either json or csv

Deprecated since version 2.6: Please use `GET /api/components/(string:project)/(string:component)/statistics/` and `GET /api/translations/(string:project)/(string:component)/(string:language)/statistics/` instead; it allows access to ACL controlled projects as well.

Retrieves statistics for given component in given format.

Example request:

```
GET /exports/stats/weblate/master/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

[
  {
    "code": "cs",
    "failing": 0,
    "failing_percent": 0.0,
    "fuzzy": 0,
    "fuzzy_percent": 0.0,
    "last_author": "Michal \u010ciha\u0159",
    "last_change": "2012-03-28T15:07:38+00:00",
    "name": "Czech",
    "total": 436,
    "total_words": 15271,
    "translated": 436,
    "translated_percent": 100.0,
    "translated_words": 3201,
    "url": "http://hosted.weblate.org/engage/weblate/cs/",
    "url_translate": "http://hosted.weblate.org/projects/weblate/master/cs/"
  },
  {
    "code": "nl",
    "failing": 21,
    "failing_percent": 4.8,
    "fuzzy": 11,
    "fuzzy_percent": 2.5,
    "last_author": null,
    "last_change": null,
    "name": "Dutch",
    "total": 436,
    "total_words": 15271,
    "translated": 319,
    "translated_percent": 73.2,
    "translated_words": 3201,
    "url": "http://hosted.weblate.org/engage/weblate/nl/",
    "url_translate": "http://hosted.weblate.org/projects/weblate/master/nl/"
  },
  {
    "code": "el",
    "failing": 11,
    "failing_percent": 2.5,
    "fuzzy": 21,
    "fuzzy_percent": 4.8,
    "last_author": null,
    "last_change": null,
    "name": "Greek",
    "total": 436,
    "total_words": 15271,
```

(continues on next page)

(continued from previous page)

```

    "translated": 312,
    "translated_percent": 71.6,
    "translated_words": 3201,
    "url": "http://hosted.weblate.org/engage/weblate/el/",
    "url_translate": "http://hosted.weblate.org/projects/weblate/master/el/"
  },
]

```

4.11.4 RSS feeds

Changes in translations are exported in RSS feeds.

GET /exports/rss/ (string: *project*) /
string: *component* / string: *language* / Retrieves RSS feed with recent changes for a translation.

GET /exports/rss/ (string: *project*) /
string: *component* / Retrieves RSS feed with recent changes for a component.

GET /exports/rss/ (string: *project*) /
 Retrieves RSS feed with recent changes for a project.

GET /exports/rss/language/ (string: *language*) /
 Retrieves RSS feed with recent changes for a language.

GET /exports/rss/
 Retrieves RSS feed with recent changes for Weblate instance.

See also:

[RSS on wikipedia](#)

4.12 Weblate Client

New in version 2.7: The `wlc` utility is fully supported since Weblate 2.7. If you are using an older version some incompatibilities with the API might occur.

4.12.1 Installation

The Weblate Client is shipped separately and includes the Python module. You need to install `wlc`; `wlc` to use these.

```
pip3 install wlc
```

4.12.2 Synopsis

```
wlc [parameter] <command> [options]
```

Commands actually indicate which operation should be performed.

4.12.3 Description

Weblate Client is Python library and command line utility to manage Weblate remotely using *Weblate's Web API*. The command line utility can be invoked as **wlc** and is built on *wlc*.

Global options

The program accepts the following global options, which must be entered before subcommand.

- format** {csv,json,text,html}
Specify output format.
- url** URL
Specify API URL. Overrides value from configuration file, see *Files*. The URL should end with `/api/`, for example `https://hosted.weblate.org/api/`.
- key** KEY
Specify API user key to use. Overrides value from configuration file, see *Files*. You can figure out your key in your profile in Weblate.
- config** PATH
Override path to configuration file, see *Files*.
- config-section** SECTION
Override section to use in configuration file, see *Files*.

Subcommands

Currently the following subcommands are available:

- version**
Prints current version.
- list-languages**
List used languages in Weblate.
- list-projects**
List projects in Weblate.
- list-components**
List components in Weblate.
- list-translations**
List translations in Weblate.
- show**
Shows Weblate object (translation, component or project).
- ls**
Lists Weblate object (translation, component or project).
- commit**
Commits changes in Weblate object (translation, component or project).
- pull**
Pulls remote repository changes into Weblate object (translation, component or project).
- push**
Pushes changes in Weblate object into remote repository (translation, component or project).

reset

New in version 0.7: Supported since wlc 0.7.

Resets changes in Weblate object to match remote repository (translation, component or project).

cleanup

New in version 0.9: Supported since wlc 0.9.

Removes any untracked changes in Weblate object to match remote repository (translation, component or project).

repo

Displays repository status for given Weblate object (translation, component or project).

statistics

Displays detailed statistics for given Weblate object (translation, component or project).

lock-status

New in version 0.5: Supported since wlc 0.5.

Displays lock status.

lock

New in version 0.5: Supported since wlc 0.5.

Locks component from translating in Weblate.

unlock

New in version 0.5: Supported since wlc 0.5.

Unlocks component from translating in Weblate.

changes

New in version 0.7: Supported since wlc 0.7 and Weblate 2.10.

Displays changes for given object.

download

New in version 0.7: Supported since wlc 0.7.

Downloads translation file.

--convert

Convert file format, if not specified not conversion happens on server and file is downloaded as is in the repository.

--output

File where to store output, if not specified file is printed to stdout.

upload

New in version 0.9: Supported since wlc 0.9.

Uploads translation file.

--overwrite

Overwrite existing translations on upload.

--input

File where to read content, if not specified file is read from stdin.

4.12.4 Files

.weblate Per project configuration file

~/.config/weblate User configuration file

/etc/xdg/weblate Global configuration file

The program follows XDG specification, so you can adjust placement of config files by environment variables `XDG_CONFIG_HOME` or `XDG_CONFIG_DIRS`.

Following settings can be configured in the `[weblate]` section (you can customize this by `--config-section`):

key

API KEY to access Weblate.

url

API server URL, defaults to `http://127.0.0.1:8000/api/`.

translation

Path of default translation, component or project.

The configuration file is INI file, for example:

```
[weblate]
url = https://hosted.weblate.org/api/
key = APIKEY
translation = weblate/master
```

Additionally API keys can be stored in the `[keys]` section:

```
[keys]
https://hosted.weblate.org/api/ = APIKEY
```

This allows you to store keys in your personal settings, while having `.weblate` configuration in the VCS repository so that `wlc` knows to which server it should talk.

4.12.5 Examples

Print current program version:

```
$ wlc version
version: 0.1
```

List all projects:

```
$ wlc list-projects
name: Hello
slug: hello
source_language: en
url: http://example.com/api/projects/hello/
web: https://weblate.org/
web_url: http://example.com/projects/hello/
```

You can also let `wlc` know current project and it will then operate on it:

```
$ cat .weblate
[weblate]
url = https://hosted.weblate.org/api/
translation = weblate/master

$ wlc show
branch: master
```

(continues on next page)

(continued from previous page)

```

file_format: po
filemask: weblate/locale/*/LC_MESSAGES/django.po
git_export: https://hosted.weblate.org/git/weblate/master/
license: GPL-3.0+
license_url: https://spdx.org/licenses/GPL-3.0+
name: master
new_base: weblate/locale/django.pot
project: weblate
repo: git://github.com/WeblateOrg/weblate.git
slug: master
template:
url: https://hosted.weblate.org/api/components/weblate/master/
vcs: git
web_url: https://hosted.weblate.org/projects/weblate/master/

```

With such setup it is easy to commit pending changes in current project:

```
$ wlc commit
```

4.13 Weblate's Python API

4.13.1 Installation

The Python API is shipped separately, you need to install *Weblate Client*:, *wlc*, to have it.

```
pip install wlc
```

4.13.2 *wlc*

WeblateException

exception *wlc.WeblateException*
Base class for all exceptions.

Weblate

class *wlc.Weblate* (*key=""*, *url=None*, *config=None*)

Parameters

- **key** (*str*) – User key
- **url** (*str*) – API server URL, if not specified default is used
- **config** (*wlc.config.WeblateConfig*) – Configuration object, overrides any other parameters.

Access class to the API, define API key and optionally API URL.

get (*path*)

Parameters **path** (*str*) – Request path

Return type *object*

Performs single API GET call.

post (*path*, ***kwargs*)

Parameters *path* (*str*) – Request path

Return type *object*

Performs single API GET call.

4.13.3 wlc.config

WeblateConfig

class `wlc.config.WeblateConfig` (*section*='wlc')

Parameters *section* (*str*) – Configuration section to use

Configuration file parser following XDG specification.

load (*path*=None)

Parameters *path* (*str*) – Path from which to load configuration.

Loads configuration from a file, if none is specified it loads from *wlc* configuration file placed in XDG configuration path (~/.config/wlc and /etc/xdg/wlc).

4.13.4 wlc.main

`wlc.main.main` (*settings*=None, *stdout*=None, *args*=None)

Parameters

- **settings** (*list*) – settings to override as list of tuples
- **stdout** (*object*) – stdout file object for printing output, uses `sys.stdout` as default
- **args** (*list*) – command line arguments to process, uses `sys.args` as default

Main entry point for command line interface.

`@wlc.main.register_command` (*command*)

Decorator to register *Command* class in main parser used by *main()*.

Command

class `wlc.main.Command` (*args*, *config*, *stdout*=None)

Main class for invoking commands.

4.14 Quick setup guide

4.14.1 Installation method

1. Docker if you are familiar with that and if you are not going to change Weblate code, see *Installing using Docker*.
2. Virtualenv, If you are not going to change Weblate code, but want to avoid Docker, see *Installing in a virtualenv*.

3. Pip, see *Installing Weblate with pip*.
4. Git, if you want to develop and/or change Weblate code, see *Installing from sources*.
5. Quick Openshift, *Installing on OpenShift 2*.

4.14.2 Installing in a virtualenv

This will create a separate Python environment for Weblate, possibly duplicating some of the Python libraries on the system.

If you'd just like to do a quick installation locally on your device to find out if Weblate is for you, you can install it using a virtual environment for Python 3, a simple (and slow!) SQLite database, and the lightweight Django development server.

1. Install the development files for libraries needed to build the Python modules:

```
# Debian/Ubuntu:
apt install libxml2-dev libxslt-dev libfreetype6-dev libjpeg-dev libz-dev libyaml-
↳dev python3-dev build-essential python3-gdbm libcairo-dev gir1.2-pango-1.0
↳libgirepository1.0-dev

# openSUSE/SLES:
zypper install libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-
↳devel libyaml-devel python3-devel cairo-devel typelib-1_0-Pango-1_0 gobject-
↳introspection-devel

# Fedora/RHEL/CentOS:
dnf install libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel
↳libyaml-devel python3-devel cairo-devel typelib-1_0-Pango-1_0 gobject-
↳introspection-devel
```

2. Install pip and virtualenv. Usually they are shipped by your distribution or with Python:

```
# Debian/Ubuntu:
apt install python3-pip python3-virtualenv virtualenv

# openSUSE/SLES:
zypper install python3-pip python3-virtualenv

# Fedora/RHEL/CentOS:
dnf install python3-pip python3-virtualenv
```

3. Create and activate the virtualenv for Weblate:

```
virtualenv --python=python3 ~/weblate-env
. ~/weblate-env/bin/activate
```

4. Activate the virtualenv for Weblate, so Weblate will look for Python libraries there first:

```
. ~/weblate-env/bin/activate
```

5. Install Weblate including all dependencies. You can also use pip to install the optional dependencies:

```
pip install Weblate
# Optional deps
pip install pytz PyYAML pyuca
# Install database backend for PostgreSQL
```

(continues on next page)

(continued from previous page)

```
pip install psycopg2-binary
# Install database backend for MySQL
apt install default-libmysqlclient-dev
pip install mysqlclient
```

6. Copy the file `~/weblate-env/lib/python3.7/site-packages/weblate/settings_example.py` to `~/weblate-env/lib/python3.7/site-packages/weblate/settings.py`
7. Optionally, adjust the values in the new `settings.py` file to your liking.
8. Create the SQLite database and its structure for Weblate:

```
weblate migrate
```

9. Create the administrator user account and copy the password it outputs to the clipboard, and also save it for later use:

```
weblate createadmin
```

10. Start the development server:

```
weblate runserver
```

11. Open a web browser, go to <http://localhost:8000/accounts/login/> and log in with the username *admin* and paste the password.
12. Proceed with [Adding translation](#) to add some translatable content to your test installation.
13. You can now run Weblate commands using **weblate** command, see [Management commands](#).

You can stop the test server with Ctrl+C, and leave the virtual environment with `deactivate`. If you want to resume testing later, you need to repeat steps 4, 8 and 11 each time to start the development server.

Note: Above described setup is useful for development, but not for production use. See [Running server](#) for detailed server setup.

4.14.3 Installing Weblate with pip

If you decide to install Weblate using the pip installer, you will notice some differences. Most importantly the command line interface is installed to the system path as **weblate** instead of `./manage.py` as used in this documentation. Also when invoking this command, you will have to specify settings by the environment variable `DJANGO_SETTINGS_MODULE` on the command line, for example:

```
DJANGO_SETTINGS_MODULE=yourproject.settings weblate migrate
```

See also:

Invoking management commands, Running server

4.14.4 Installing from sources

1. Grab the latest Weblate sources using Git (or download a tarball and unpack that):

```
git clone https://github.com/WeblateOrg/weblate.git
```

Alternatively you can use released archives. You can download them from our website <<https://weblate.org/>>. Those downloads are cryptographically signed, please see [Verifying release signatures](#).

Note: If you are running a version from Git, you should also regenerate locale files every time you are upgrading. You can do this by invoking the script `./scripts/generate-locales`.

2. Install all required dependencies into an virtual env (also see [Software requirements](#)):

```
virtualenv --python=python3 .venv
. .venv/bin/activate
pip install -r /path/to/weblate/requirements.txt
```

3. Copy `weblate/settings_example.py` to `weblate/settings.py` and adjust it to match your setup. You will at least need to configure the database connection (possibly adding a user and creating the database). Check [Configuration](#) for Weblate specific configuration options.
4. Create the database used by Weblate, see [Database setup for Weblate](#).
5. Build Django tables, static files and initial data (see [Filling up the database](#) and [Serving static files](#)):

```
./manage.py migrate
./manage.py collectstatic
./scripts/generate-locales # If you are using Git checkout
```

6. Configure webserver to serve Weblate, see [Running server](#).

Note: Running latest version from Git `master` branch should be safe. It is maintained, stable and production ready. It is most often the version running [Hosted Weblate](#).

See also:

[Running server](#)

4.14.5 Installing using Docker

1. Clone `weblate-docker` repo:

```
git clone https://github.com/WeblateOrg/docker-compose.git weblate-docker
cd weblate-docker
```

2. Start Weblate containers:

```
docker-compose up
```

See also:

See [Running Weblate with Docker](#) for more detailed instructions and customization options.

4.14.6 Installing on OpenShift 2

1. You can install Weblate on OpenShift PaaS directly from its Git repository using the OpenShift Client Tools:

```
rhc -awebplate app create -t python-2.7 --from-code https://github.com/
↪WeblateOrg/weblate.git --no-git
```

2. After installation everything should be preconfigured, and you can immediately start adding a translation project as described below.

See also:

For more info, including how to retrieve the generated admin password, see [Running Weblate on OpenShift 2](#).

4.14.7 Adding translation

1. Open the admin interface (<http://localhost/admin/>) and create the project you want to translate. See [Project configuration](#) for more details.

All you need to specify here is the project name and its website.

2. Create a component which is the real object for translation - it points to the VCS repository, and selects which files to translate. See [Component configuration](#) for more details.

The important fields here are: Component name, VCS repository address and mask for finding translatable files. Weblate supports a wide range of formats including gettext PO files, Android resource strings, iOS string properties, Java properties or Qt Linguist files, see [Supported file formats](#) for more details.

3. Once the above is completed (it can be lengthy process depending on the size of your VCS repository, and number of messages to translate), you can start translating.

4.14.8 Installing requirements

Weblate can be also installed to use system packages when available. The following guides should give you guidance to do that, but the exact setup depends on distribution version you are using.

Requirements on Debian or Ubuntu

On recent releases of Debian or Ubuntu, most of the requirements are already packaged, to install them you can use apt:

```
apt install python3-pip python3-django translate-toolkit \
python3-whoosh python3-pil \
git mercurial \
python3-django-compressor python3-django-crispy-forms \
python3-djangorestframework python3-dateutil python3-celery \
python3-gdbm

# Optional packages for database backend:

# For PostgreSQL
apt install python3-psycopg2
# For MySQL on Ubuntu (if using the Ubuntu package for Django)
apt install python3-pymysql
# For MySQL on Debian (or Ubuntu if using upstream Django packages)
apt install python3-mysqldb
```

On older releases, some required dependencies are missing or outdated, so you need to install several Python modules manually using pip:


```
# Dependencies for ``python-social-auth``
apt install python3-requests-oauthlib python3-six python3-openid

# Social auth
pip install social-auth-core
pip install social-auth-app-django

# In case your distribution has ``python-django`` older than 1.9
pip install Django

# In case the ``python-django-crispy-forms`` package is missing
pip install django-crispy-forms

# In case ``python-whoosh`` package is missing or older than 2.7
pip install whoosh

# In case the ``python-django-compressor`` package is missing,
# Try installing it by its older name, or by using pip:
apt install python3-compressor
pip install django_compressor

# Optional for OCR support
apt install tesseract-ocr libtesseract-dev liblibleptonica-dev cython
pip install tesseractocr

# Install database backend for PostgreSQL
pip install psycopg2-binary
# Install database backend for MySQL
apt install default-libmysqlclient-dev
pip install mysqlclient
```

For proper sorting of Unicode strings, it is recommended to install pyuca:

```
pip install pyuca
```

Depending on how you intend to run Weblate and what you already have installed, you might need additional components:

```
# Web server option 1: NGINX and uWSGI
apt install nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
apt install apache2 libapache2-mod-wsgi

# Caching backend: Redis
apt install redis-server

# Database option 1: PostgreSQL
apt install postgresql

# Database option 2: MariaDB
apt install mariadb-server

# Database option 3: MySQL
apt install mysql-server

# SMTP server
```

(continues on next page)

(continued from previous page)

```
apt install exim4

# GitHub PR support: ``hub``
# See https://hub.github.com/
```

Requirements on openSUSE

Most of requirements are available either directly in openSUSE or in `devel:languages:python` repository:

```
zypper install python3-Django translate-toolkit \
python3-Whoosh python3-Pillow \
python3-social-auth-core python3-social-auth-app-django \
Git mercurial python3-pyuca \
python3-dateutil python3-celery

# Optional for database backend
zypper install python3-psycopg2      # For PostgreSQL
zypper install python3-MySQL-python # For MySQL
```

Depending on how you intend to run Weblate and what you already have installed, you might need additional components:

```
# Web server option 1: NGINX and uWSGI
zypper install nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
zypper install apache2 apache2-mod_wsgi

# Caching backend: Redis
zypper install redis-server

# Database option 1: PostgreSQL
zypper install postgresql

# Database option 2: MariaDB
zypper install mariadb

# Database option 3: MySQL
zypper install mysql

# SMTP server
zypper install postfix

# GitHub PR support: ``hub``
# See https://hub.github.com/
```

Requirements on macOS

If your Python was not installed using `brew`, make sure you have this in your `.bash_profile` file or executed somehow:

```
export PYTHONPATH="/usr/local/lib/python3.7/site-packages:$PYTHONPATH"
```

This configuration makes the installed libraries available to Python.

Requirements using pip installer

Most requirements can be also installed using the pip installer:

```
pip install -r requirements.txt
```

For building some of the extensions development files for several libraries are required, see *Installing in a virtualenv* for instructions how to install these.

All optional dependencies (see above) can be installed using:

```
pip install -r requirements-optional.txt
```

4.14.9 Verifying release signatures

Weblate release are cryptographically signed by the releasing developer. Currently this is Michal Čihař. Fingerprint of his PGP key is:

```
63CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D
```

and you can get more identification information from <https://keybase.io/nijel>.

You should verify that the signature matches the archive you have downloaded. This way you can be sure that you are using the same code that was released. You should also verify the date of the signature to make sure that you downloaded the latest version.

Each archive is accompanied with .asc files which contains the PGP signature for it. Once you have both of them in the same folder, you can verify the signature:

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Ne 3. března 2019, 16:43:15 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Can't check signature: public key not found
```

As you can see gpg complains that it does not know the public key. At this point you should do one of the following steps:

- Use wkd to download the key:

```
$ gpg --auto-key-locate wkd --locate-keys michal@cihar.com
pub  rsa4096 2009-06-17 [SC]
    63CB1DF1EF12CF2AC0EE5A329C27B31342B7511D
uid          [ultimate] Michal Čihař <michal@cihar.com>
uid          [ultimate] Michal Čihař <nijel@debian.org>
uid          [ultimate] [jpeg image of size 8848]
uid          [ultimate] Michal Čihař (Brains) <michal.cihar@brains.cz>
sub  rsa4096 2009-06-17 [E]
sub  rsa4096 2015-09-09 [S]
```

- Download the keyring from [Michal's server](#), then import it with:

```
$ gpg --import wmxth3chu9jfxdxywjlskpmhsj31lmzm
```

- Download and import the key from one of the key servers:

```
$ gpg --keyserver hkp://pgp.mit.edu --recv-keys 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: key 9C27B31342B7511D: "Michal Čihař <michal@cihar.com>" imported
gpg: Total number processed: 1
gpg:                unchanged: 1
```

This will improve the situation a bit - at this point you can verify that the signature from the given key is correct but you still can not trust the name used in the key:

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Ne 3. března 2019, 16:43:15 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Good signature from "Michal Čihař <michal@cihar.com>" [ultimate]
gpg:                aka "Michal Čihař <nijel@debian.org>" [ultimate]
gpg:                aka "[jpeg image of size 8848]" [ultimate]
gpg:                aka "Michal Čihař (Brains) <michal.cihar@braiins.cz>" [ultimate]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                There is no indication that the signature belongs to the owner.
Primary key fingerprint: 63CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D
```

The problem here is that anybody could issue the key with this name. You need to ensure that the key is actually owned by the mentioned person. The GNU Privacy Handbook covers this topic in the chapter [Validating other keys on your public keyring](#). The most reliable method is to meet the developer in person and exchange key fingerprints, however you can also rely on the web of trust. This way you can trust the key transitively though signatures of others, who have met the developer in person.

Once the key is trusted, the warning will not occur:

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Sun Mar  3 16:43:15 2019 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Good signature from "Michal Čihař <michal@cihar.com>" [ultimate]
gpg:                aka "Michal Čihař <nijel@debian.org>" [ultimate]
gpg:                aka "[jpeg image of size 8848]" [ultimate]
gpg:                aka "Michal Čihař (Brains) <michal.cihar@braiins.cz>" [ultimate]
```

Should the signature be invalid (the archive has been changed), you would get a clear error regardless of the fact that the key is trusted or not:

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: Signature made Sun Mar  3 16:43:15 2019 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: BAD signature from "Michal Čihař <michal@cihar.com>" [ultimate]
```

4.15 Installation instructions

Looking for quick installation instructions? See [Quick setup guide](#).

4.15.1 Hardware requirements

Weblate should run on all contemporary hardware without problems, the following is the minimal configuration required to run Weblate on a single host (Weblate, database and webserver):

- 2 GB of RAM
- 2 CPU cores
- 1 GB of storage space

The more memory the better - it is used for caching on all levels (filesystem, database and Weblate).

Many concurrent users increases the amount of needed CPU cores. For hundreds of translation components at least 4 GB of RAM is recommended.

Note: Actual requirements for your installation of Weblate vary heavily based on the size of the translations managed in it.

4.15.2 Software requirements

Python dependencies

Weblate is written in [Python](#) and supports Python 2.7, 3.4 or newer. You can install dependencies using pip or from your distribution packages, full list of them is available in `requirements.txt`.

Most notable dependencies:

Django <https://www.djangoproject.com/>

Celery <http://www.celeryproject.org/>

Translate Toolkit <https://toolkit.translatehouse.org/>

translation-finder <https://github.com/WeblateOrg/translation-finder>

Python Social Auth <https://python-social-auth.readthedocs.io/>

Whoosh <https://bitbucket.org/mchaput/whoosh/wiki/Home>

Django REST Framework <https://www.django-rest-framework.org/>

Optional dependencies

Following modules are necessary for some of Weblate features. You can find all of them in `requirements-optional.txt`.

Mercurial (optional for Mercurial repositories support) <https://www.mercurial-scm.org/>

pyuca (optional for proper sorting of strings) <https://github.com/jtauber/pyuca>

phply (optional for PHP support) <https://github.com/viraptor/phply>

tesseract (optional for screenshots OCR) <https://github.com/sirfz/tesseract>

akismet (optional for suggestion spam protection) <https://github.com/ubernostrum/akismet>

ruamel.yaml (optional for *YAML files*) <https://pypi.org/project/ruamel.yaml/>

backports.csv (needed on Python 2.7) <https://pypi.org/project/backports.csv/>

Zeep (optional for *Microsoft Terminology Service*) <https://python-zeep.readthedocs.io/>

aeidon (optional for *Subtitle files*) <https://pypi.org/project/aeidon/>

Database backend dependencies

Any database supported in Django will work, see *Database setup for Weblate* and backends documentation for more details.

Other system requirements

The following dependencies have to be installed on the system:

Git <https://git-scm.com/>

Pango, Cairo and related header files and gir introspection data <https://cairographics.org/>, <https://www.pango.org/>, see *Pango and Cairo*

hub (optional for sending pull requests to GitHub) <https://hub.github.com/>

git-review (optional for Gerrit support) <https://pypi.org/project/git-review/>

git-svn (optional for Subversion support) <https://git-scm.com/docs/git-svn>

tesseract and it's data (optional for screenshots OCR) <https://github.com/tesseract-ocr/tesseract>

Compile time dependencies

To compile some of the *Python dependencies* you might need to install their dependencies. This depends on how you install them, so please consult individual packages for documentation. You won't need those if using prebuilt wheels while installing using `pip` or when you use distribution packages.

Pango and Cairo

Changed in version 3.7.

Weblate uses Pango and Cairo for rendering bitmap widgets (see *Promoting the translation*) and rendering checks (see *Managing fonts*). To properly install Python bindings for those you need to install system libraries first - you need both Cairo and Pango, which in turn need Glib. All those should be installed with development files and GObject introspection data.

4.15.3 Installing Weblate

Choose an installation method that best fits your environment in our *Quick setup guide*.

4.15.4 Filesystem permissions

The Weblate process needs to be able to read and write to the directory where it keeps data - `DATA_DIR`.

The default configuration places them in the same tree as the Weblate sources, however you might prefer to move these to a better location such as: `/var/lib/weblate`.

Weblate tries to create these directories automatically, but it will fail when it does not have permissions to do so.

You should also take care when running *Management commands*, as they should be ran under the same user as Weblate itself is running, otherwise permissions on some files might be wrong.

See also:

Serving static files

4.15.5 Database setup for Weblate

It is recommended to run Weblate with PostgreSQL database server. Using a SQLite backend is really only suitable for testing purposes.

See also:

Use a powerful database engine, [Databases](#)

PostgreSQL

PostgreSQL is usually the best choice for Django based sites. It's the reference database used for implementing Django database layer.

See also:

[PostgreSQL notes](#)

Creating a database in PostgreSQL

It is usually a good idea to run Weblate in a separate database, and separate user account:

```
# If PostgreSQL was not installed before, set the master password
sudo -u postgres psql postgres -c "\password postgres"

# Create a database user called "weblate"
sudo -u postgres createuser -D -P weblate

# Create the database "weblate" owned by "weblate"
sudo -u postgres createdb -O weblate weblate
```

Configuring Weblate to use PostgreSQL

The `settings.py` snippet for PostgreSQL:

```
DATABASES = {
    'default': {
        # Database engine
        'ENGINE': 'django.db.backends.postgresql',
        # Database name
        'NAME': 'weblate',
        # Database user
        'USER': 'weblate',
        # Database password
        'PASSWORD': 'password',
        # Set to empty string for localhost
        'HOST': 'database.example.com',
        # Set to empty string for default
        'PORT': '',
    }
}
```

MySQL or MariaDB

MySQL or MariaDB are quite good choices for running Weblate. However when using MySQL you might hit some problems caused by it.

Warning: It's likely that MySQL/MariaDB support will be dropped in future Weblate releases, so it's not recommended for new installations.

See also:

[MySQL notes](#)

Unicode issues in MySQL

MySQL by default uses something called `utf8`, which can not store all Unicode characters, only those who fit into three bytes in `utf-8` encoding. In case you're using emojis or some other higher Unicode symbols you might hit errors when saving such data. Depending on the MySQL and Python bindings version, the produced error might look like this:

- *OperationalError: (1366, "Incorrect string value: '\xF0xA8xABxA1' for column 'target' at row 1")*
- *UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-3: ordinal not in range(128)*

To solve this, you need to change your database to use `utf8mb4` (which is again a subset of Unicode, but this time one which can be stored in four bytes in `utf-8` encoding, thus covering all chars currently defined in Unicode).

This can be achieved during creation of the database by selecting this character set (see [Creating a database in MySQL](#)) and specifying that character set in the connection settings (see [Configuring Weblate to use MySQL](#)).

In case you have an existing database, you can change it to `utf8mb4` by, but this won't change collation of existing fields:

```
ALTER DATABASE weblate CHARACTER SET utf8mb4;
```

Using this charset might however lead to problems with the default MySQL server settings, as each character now takes 4 bytes to store, and MySQL has an upper limit of 767 bytes for an index. In case this happens you will get one of the following error messages:

- *1071: Specified key was too long; max key length is 767 bytes*
- *1709: Index column size too large. The maximum column size is 767 bytes.*

There are two ways to work around this limitation. You can configure MySQL to not have this limit, see [Using InnoDB_large_prefix to Avoid ERROR 1071](#). Alternatively you can also adjust several settings for social-auth in your `settings.py` (see [Configuration](#)):

```
# Limit some social-auth fields to 191 chars to fit
# them in 767 bytes

SOCIAL_AUTH_UID_LENGTH = 191
SOCIAL_AUTH_NONCE_SERVER_URL_LENGTH = 191
SOCIAL_AUTH_ASSOCIATION_SERVER_URL_LENGTH = 191
SOCIAL_AUTH_ASSOCIATION_HANDLE_LENGTH = 191
SOCIAL_AUTH_EMAIL_LENGTH = 191
```


Transaction locking

MySQL by default uses a different transaction locking scheme than other databases, and in case you see errors like *Deadlock found when trying to get lock; try restarting transaction* it might be good idea to enable *STRICT_TRANS_TABLES* mode in MySQL. This can be done in the server configuration file (usually `/etc/mysql/my.cnf` on Linux):

```
[mysqld]
sql-mode=STRICT_TRANS_TABLES
```

See also:

Setting `sql_mode`

Creating a database in MySQL

Create weblate user to access the weblate database:

```
# Grant all privileges to the user ``weblate``
GRANT ALL PRIVILEGES ON weblate.* TO 'weblate'@'localhost' IDENTIFIED BY 'password';

# Create a database on MySQL >= 5.7.7
CREATE DATABASE weblate CHARACTER SET utf8mb4;

# Use utf8 for older versions
# CREATE DATABASE weblate CHARACTER SET utf8;
```

Configuring Weblate to use MySQL

The `settings.py` snippet for MySQL:

```
DATABASES = {
    'default': {
        # Database engine
        'ENGINE': 'django.db.backends.mysql',
        # Database name
        'NAME': 'weblate',
        # Database user
        'USER': 'weblate',
        # Database password
        'PASSWORD': 'password',
        # Set to empty string for localhost
        'HOST': 'database.example.com',
        # Set to empty string for default
        'PORT': '',
        # Additional database options
        'OPTIONS': {
            # In case of using an older MySQL server, which has MyISAM as a default_
↪ storage
            # 'init_command': 'SET storage_engine=INNODB',
            # Uncomment for MySQL older than 5.7:
            # 'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
            # If your server supports it, see the Unicode issues above
            'charset': 'utf8mb4',
```

(continues on next page)

(continued from previous page)

```
}  
  
}  
  
}
```

4.15.6 Other configurations

Configuring outgoing email

Weblate sends out emails on various occasions - for account activation and on various notifications configured by users. For this it needs access to a SMTP server.

The mail server setup is configured using these settings: `EMAIL_HOST`, `EMAIL_HOST_PASSWORD`, `EMAIL_HOST_USER` and `EMAIL_PORT`. Their names are quite self-explanatory, but you can find more info in the Django documentation.

Note: You can verify whether outgoing email is working correctly by using the `sendtestemail` management command (see *Invoking management commands* for instructions how to invoke it in different environments).

HTTP proxy

Weblate does execute VCS commands and those accept proxy configuration from environment. The recommended approach is to define proxy settings in `settings.py`:

```
import os  
os.environ['http_proxy'] = "http://proxy.example.com:8080"  
os.environ['HTTPS_PROXY'] = "http://proxy.example.com:8080"
```

See also:

[Proxy Environment Variables](#)

4.15.7 Installation

See also:

Sample configuration

Copy `weblate/settings_example.py` to `weblate/settings.py` and adjust it to match your setup. You will probably want to adjust the following options: `ADMINS`

List of site administrators to receive notifications when something goes wrong, for example notifications on failed merges, or Django errors.

See also:

`ADMINS`

`ALLOWED_HOSTS`

If you are running Django 1.5 or newer, you need to set this to list the hosts your site is supposed to serve. For example:

```
ALLOWED_HOSTS = ['demo.weblate.org']
```

See also:

`ALLOWED_HOSTS`

`SESSION_ENGINE`

Configure how your sessions will be stored. In case you keep the default database backend engine, you should schedule: `./manage.py clearsessions` to remove stale session data from the database.

If you are using Redis as cache (see [Enable caching](#)) it is recommended to use it for sessions as well:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
```

See also:

[Configuring the session engine](#), `SESSION_ENGINE`

`DATABASES`

Connectivity to database server, please check Django's documentation for more details.

See also:

[Database setup for Weblate](#), `DATABASES`, [Databases](#)

`DEBUG`

Disable this for any production server. With debug mode enabled, Django will show backtraces in case of error to users, when you disable it, errors will be sent per email to ADMINS (see above).

Debug mode also slows down Weblate, as Django stores much more info internally in this case.

See also:

`DEBUG`,

`DEFAULT_FROM_EMAIL`

Email sender address for outgoing email, for example registration emails.

See also:

`DEFAULT_FROM_EMAIL`,

`SECRET_KEY`

Key used by Django to sign some info in cookies, see [Django secret key](#) for more info.

`SERVER_EMAIL`

Email used as sender address for sending emails to the administrator, for example notifications on failed merges.

See also:

`SERVER_EMAIL`

4.15.8 Filling up the database

After your configuration is ready, you can run `./manage.py migrate` to create the database structure. Now you should be able to create translation projects using the admin interface.

In case you want to run an installation non interactively, you can use `./manage.py migrate --noinput`, and then create an admin user using `createadmin` command.

You should also log in to the admin interface (on `/admin/` URL) and adjust the default sitename to match your domain by clicking on *Sites* and once there, change the `example.com` record to match your real domain name.

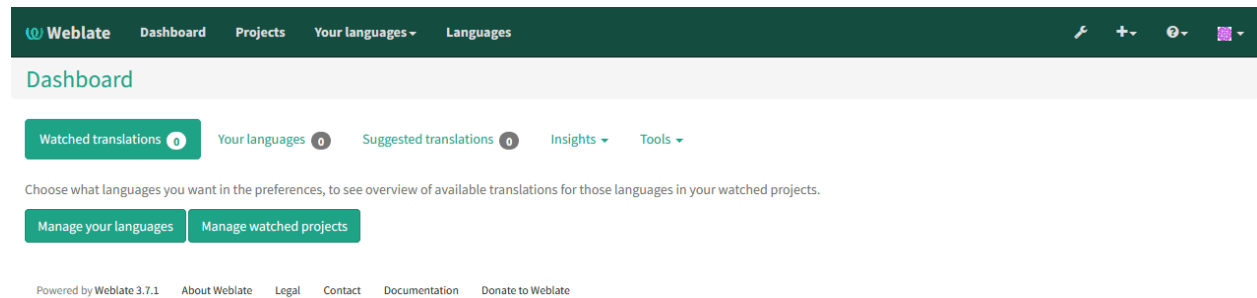
Once you are done, you should also check the *Performance report* in the admin interface, which will give you hints of potential non optimal configuration on your site.

See also:

Configuration, Access control, Why do links contain example.com as the domain?, Set correct sitename

4.15.9 Production setup

For a production setup you should carry out adjustments described in the following sections. The most critical settings will trigger a warning, which is indicated by a red exclamation mark in the top bar if logged in as a superuser:



It is also recommended to inspect checks triggered by Django (though you might not need to fix all of them):

```
./manage.py check --deploy
```

See also:

[Deployment checklist](#)

Disable debug mode

Disable Django's debug mode (`DEBUG`) by:

```
DEBUG = False
```

With debug mode on, Django stores all executed queries and shows users backtraces of errors, which is not desired in a production setup.

See also:

Installation

Properly configure admins

Set the correct admin addresses to the `ADMINS` setting to defining who will receive emails in case something goes wrong on the server, for example:

```
ADMINS = (
    ('Your Name', 'your_email@example.com'),
)
```

See also:*Installation***Set correct sitename**

Adjust sitename in the admin interface, otherwise links in RSS or registration emails will not work.

Please open the admin interface and edit the default sitename and domain under the *Sites > Sites* (or do it directly at the `/admin/sites/site/1/` URL under your Weblate installation). You have to change the *Domain name* to match your setup.

Note: This setting should only contain the domain name. For configuring protocol, (enabling HTTPS) use `ENABLE_HTTPS` and for changing URL, use `URL_PREFIX`.

Alternatively, you can set the site name from the commandline using `changesite`. For example, when using the built-in server:

```
./manage.py changesite --set-name 127.0.0.1:8000
```

For a production site, you want something like:

```
./manage.py changesite --set-name weblate.example.com
```

See also:*Why do links contain example.com as the domain?, changesite, The “sites” framework***Correctly configure HTTPS**

It is strongly recommended to run Weblate using the encrypted HTTPS protocol. After enabling it, you should set `ENABLE_HTTPS` in the settings, which also adjusts several other related Django settings in the example configuration.

You might want to set up HSTS as well, see [SSL/HTTPS](#) for more details.

Use a powerful database engine

Please use PostgreSQL for a production environment, see *Database setup for Weblate* for more info.

See also:*Database setup for Weblate, Installation, Databases***Enable caching**

If possible, use Redis from Django by adjusting the `CACHES` configuration variable, for example:

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/0',
        # If redis is running on same host as Weblate, you might
        # want to use unix sockets instead:
        # 'LOCATION': 'unix:///var/run/redis/redis.sock?db=0',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
            'PARSER_CLASS': 'redis.connection.HiredisParser',
        }
    }
}
```

See also:

Avatar caching, Django's cache framework

Avatar caching

In addition to caching of Django, Weblate performs caching of avatars. It is recommended to use a separate, file-backed cache for this purpose:

```
CACHES = {
    'default': {
        # Default caching backend setup, see above
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'unix:///var/run/redis/redis.sock?db=0',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
            'PARSER_CLASS': 'redis.connection.HiredisParser',
        }
    },
    'avatar': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': os.path.join(DATA_DIR, 'avatar-cache'),
        'TIMEOUT': 604800,
        'OPTIONS': {
            'MAX_ENTRIES': 1000,
        }
    }
}
```

See also:

ENABLE_AVATARS, *AVATAR_URL_PREFIX*, *Avatars*, *Enable caching*, Django's cache framework

Configure email addresses

Weblate needs to send out emails on several occasions, and these emails should have a correct sender address, please configure *SERVER_EMAIL* and *DEFAULT_FROM_EMAIL* to match your environment, for example:

```
SERVER_EMAIL = 'admin@example.org'
DEFAULT_FROM_EMAIL = 'weblate@example.org'
```

See also:

Installation, *Configuring outgoing email*, *DEFAULT_FROM_EMAIL*, *SERVER_EMAIL*

Allowed hosts setup

Django 1.5 and newer require `ALLOWED_HOSTS` to hold a list of domain names your site is allowed to serve, leaving it empty will block any requests.

See also:

`ALLOWED_HOSTS`

The `pyuca` library

The `pyuca` library is optionally used by Weblate to sort Unicode strings. This way language names are properly sorted, even in non-ASCII languages like Japanese, Chinese or Arabic, or for languages that employ [diacritics](#).

Django secret key

The `SECRET_KEY` setting is used by Django to sign cookies, and you should really generate your own value rather than using the one from the example setup.

You can generate a new key using `weblate/examples/generate-secret-key` shipped with Weblate.

See also:

`SECRET_KEY`

Home directory

Changed in version 2.1: This is no longer required, Weblate now stores all its data in `DATA_DIR`.

The home directory for the user running Weblate should exist and be writable by this user. This is especially needed if you want to use SSH to access private repositories, but Git might need to access this directory as well (depending on the Git version you use).

You can change the directory used by Weblate in `settings.py`, for example to set it to `configuration` directory under the Weblate tree:

```
os.environ['HOME'] = os.path.join(BASE_DIR, 'configuration')
```

Note: On Linux, and other UNIX like systems, the path to user's home directory is defined in `/etc/passwd`. Many distributions default to a non-writable directory for users used for serving web content (such as `apache`, `www-data` or `wwwrun`, so you either have to run Weblate under a different user, or change this setting.

See also:

Accessing repositories

Template loading

It is recommended to use a cached template loader for Django. It caches parsed templates and avoids the need to do parsing with every single request. You can configure it using the following snippet (the `loaders` setting is important here):

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates'),
        ],
        'OPTIONS': {
            'context_processors': [
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.debug',
                'django.template.context_processors.i18n',
                'django.template.context_processors.request',
                'django.template.context_processors.csrf',
                'django.contrib.messages.context_processors.messages',
                'weblate.trans.context_processors.weblate_context',
            ],
            'loaders': [
                ('django.template.loaders.cached.Loader', [
                    'django.template.loaders.filesystem.Loader',
                    'django.template.loaders.app_directories.Loader',
                ]),
            ],
        },
    ],
]
```

See also:

`django.template.loaders.cached.Loader`

Running maintenance tasks

For optimal performance, it is good idea to run some maintenance tasks in the background. This is now automatically done by *Background tasks using Celery* and covers following tasks:

- Configuration health check (hourly).
- Committing pending changes (hourly), see *Lazy commits* and *commit_pending*.
- Updating component alerts (daily).
- Update remote branches (nightly), see *AUTO_UPDATE*.
- Translation memory backup to JSON (daily), see *dump_memory*.
- Fulltext and database maintenance tasks (daily and weekly taks), see *cleanup_trans*.

Changed in version 3.2: Since version 3.2, the default way of executing these tasks is using Celery and Weblate already comes with proper configuration, see *Background tasks using Celery*.

4.15.10 Running server

You will need several services to run Weblate, the recommended setup consists of:

- Database server (see *Database setup for Weblate*)
- Cache server (see *Enable caching*)
- Frontend web server for static files and SSL termination (see *Serving static files*)

- Wsgi server for dynamic content (see *Sample configuration for NGINX and uWSGI*)
- Celery for executing background tasks (see *Background tasks using Celery*)

Note: There are some dependencies between the services, for example cache and database should be running when starting up Celery or uwsgi processes.

In most cases, you will run all services on single (virtual) server, but in case your installation is heavy loaded, you can split up the services. The only limitation on this is that Celery and Wsgi servers need access to `DATA_DIR`.

Running web server

Running Weblate is not different from running any other Django based program. Django is usually executed as uWSGI or fcgi (see examples for different webserver below).

For testing purposes, you can use the built-in web server in Django:

```
./manage.py runserver
```

Warning: Do not use this in production, as this has severe performance limitations.

Serving static files

Changed in version 2.4: Prior to version 2.4, Weblate didn't properly use the Django static files framework and the setup was more complex.

Django needs to collect its static files in a single directory. To do so, execute `./manage.py collectstatic --noinput`. This will copy the static files into a directory specified by the `STATIC_ROOT` setting (this defaults to a `static` directory inside `DATA_DIR`).

It is recommended to serve static files directly from your web server, you should use that for the following paths:

/static/ Serves static files for Weblate and the admin interface (from defined by `STATIC_ROOT`).

/media/ Used for user media uploads (e.g. screenshots).

/favicon.ico Should be rewritten to rewrite a rule to serve `/static/favicon.ico`

/robots.txt Should be rewritten to rewrite a rule to serve `/static/robots.txt`

See also:

[Deploying Django](#), [Deploying static files](#)

Content security policy

The default Weblate configuration enables `weblate.middleware.SecurityMiddleware` middleware which sets security related HTTP headers like `Content-Security-Policy` or `X-XSS-Protection`. These are by default set up to work with Weblate and it's configuration, but this might clash with your customization. If that is the case, it is recommended to disable this middleware and set these headers manually.

Sample configuration for Apache

The following configuration runs Weblate as WSGI, you need to have enabled `mod_wsgi` (available as `weblate/examples/apache.conf`):

```
#
# VirtualHost for weblate
#
# This example assumes Weblate is installed in /usr/share/weblate
#
# If using virtualenv, you need to add it to search path as well:
# WSGIPythonPath /usr/share/weblate:/path/to/your/venv/lib/python3.7/site-packages
#
<VirtualHost *:80>
    ServerAdmin admin@weblate.example.org
    ServerName weblate.example.org

    # DATA_DIR/static/robots.txt
    Alias /robots.txt /var/lib/weblate/static/robots.txt
    # DATA_DIR/static/favicon.ico
    Alias /favicon.ico /var/lib/weblate/static/favicon.ico

    # DATA_DIR/static/
    Alias /static/ /var/lib/weblate/static/
    <Directory /var/lib/weblate/static/>
        Require all granted
    </Directory>

    # DATA_DIR/media/
    Alias /media/ /var/lib/weblate/media/
    <Directory /var/lib/weblate/media/>
        Require all granted
    </Directory>

    WSGIDaemonProcess weblate.example.org python-path=/usr/share/weblate
    WSGIProcessGroup weblate.example.org
    WSGIApplicationGroup %{GLOBAL}

    WSGIScriptAlias / /usr/share/weblate/weblate/wsgi.py process-group=weblate.
    ↪example.org
    WSGIPassAuthorization On

    <Directory /usr/share/weblate/weblate>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
</VirtualHost>
```

This configuration is for Apache 2.4 and later. For earlier versions of Apache, replace *Require all granted* with *Allow from all*.

See also:

How to use Django with Apache and `mod_wsgi`

Sample configuration for Apache and Gunicorn

The following configuration runs Weblate in Gunicorn and Apache 2.4 (available as `weblate/examples/apache.gunicorn.conf`):

```
#
# VirtualHost for weblate using gunicorn on localhost:8000
#
# This example assumes Weblate is installed in /usr/share/weblate
#
#
<VirtualHost *:443>
    ServerAdmin admin@weblate.example.org
    ServerName weblate.example.org

    # DATA_DIR/static/robots.txt
    Alias /robots.txt /var/lib/weblate/static/robots.txt
    # DATA_DIR/static/favicon.ico
    Alias /favicon.ico /var/lib/weblate/static/favicon.ico

    # DATA_DIR/static/
    Alias /static/ /var/lib/weblate/static/
    <Directory /var/lib/weblate/static/>
        Require all granted
    </Directory>

    # DATA_DIR/media/
    Alias /media/ /var/lib/weblate/media/
    <Directory /var/lib/weblate/media/>
        Require all granted
    </Directory>

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/https_cert.cert
    SSLCertificateKeyFile /etc/apache2/ssl/https_key.pem
    SSLProxyEngine On

    ProxyPass /robots.txt !
    ProxyPass /favicon.ico !
    ProxyPass /static/ !
    ProxyPass /media/ !

    ProxyPass / http://localhost:8000/
    ProxyPassReverse / http://localhost:8000/
    ProxyPreserveHost On
</VirtualHost>
```

See also:

[How to use Django with Gunicorn](#)

Sample configuration for NGINX and uWSGI

To run production webserver, use the wsgi wrapper installed with Weblate (in virtual env case it is installed as `~/weblate-env/lib/python3.7/site-packages/weblate/wsgi.py`). Don't forget to set the Python search path to your virtualenv as well (for example using `virtualenv = /home/user/weblate-env` in

uWSGI).

The following configuration runs Weblate as uWSGI under the NGINX webserver.

Configuration for NGINX (also available as `weblate/examples/weblate.nginx.conf`):

```
server {
    listen 80;
    server_name weblate;
    root /usr/share/weblate;

    location ~ ^/favicon.ico$ {
        # DATA_DIR/static/favicon.ico
        alias /var/lib/weblate/static/favicon.ico;
        expires 30d;
    }

    location ~ ^/robots.txt$ {
        # DATA_DIR/static/robots.txt
        alias /var/lib/weblate/static/robots.txt;
        expires 30d;
    }

    location /static/ {
        # DATA_DIR/static/
        alias /var/lib/weblate/static/;
        expires 30d;
    }

    location /media/ {
        # DATA_DIR/media/
        alias /var/lib/weblate/media/;
        expires 30d;
    }

    location / {
        include uwsgi_params;
        # Needed for long running operations in admin interface
        uwsgi_read_timeout 3600;
        # Adjust based to uwsgi configuration:
        uwsgi_pass unix:///run/uwsgi/app/weblate/socket;
        # uwsgi_pass 127.0.0.1:8080;
    }
}
```

Configuration for uWSGI (also available as `weblate/examples/weblate.uwsgi.ini`):

```
[uwsgi]
plugins      = python
master       = true
protocol     = uwsgi
socket       = 127.0.0.1:8080
wsgi-file    = /usr/local/lib/python3.6/dist-packages/weblate/wsgi.py

# Add path to Weblate checkout if you did not install
# Weblate by pip
# python-path = /path/to/weblate

# In case you're using virtualenv uncomment this:
```

(continues on next page)

(continued from previous page)

```
# virtualenv = /path/to/weblate/virtualenv

# Needed for OAuth/OpenID
buffer-size = 8192

# Increase number of workers for heavily loaded sites
# workers = 6

# Child processes do not need file descriptors
close-on-exec = true

# Avoid default 0000 umask
umask = 0022

# Run as weblate user
uid = weblate
gid = weblate

# Enable harakiri mode (kill requests after some time)
# harakiri = 3600
# harakiri-verbose = true

# Enable uWSGI stats server
# stats = :1717
# stats-http = true

# Do not log some errors caused by client disconnects
ignore-sigpipe = true
ignore-write-errors = true
disable-write-exception = true
```

See also:

How to use Django with uWSGI

Running Weblate under path

Changed in version 1.3: This is supported since Weblate 1.3.

A sample Apache configuration to serve Weblate under /weblate. Again using mod_wsgi (also available as weblate/examples/apache-path.conf):

```
# Example Apache configuration for running Weblate under /weblate path

WSGIScriptAlias /usr/share/weblate
# If using virtualenv, you need to add it to search path as well:
# WSGIScriptAlias /usr/share/weblate:/path/to/your/venv/lib/python3.7/site-packages
<VirtualHost *:80>
    ServerAdmin admin@image.weblate.org
    ServerName image.weblate.org

    # DATA_DIR/static/robots.txt
    Alias /weblate/robots.txt /var/lib/weblate/static/robots.txt
    # DATA_DIR/static/favicon.ico
    Alias /weblate/favicon.ico /var/lib/weblate/static/favicon.ico
```

(continues on next page)

(continued from previous page)

```

# DATA_DIR/static/
Alias /weblate/static/ /var/lib/weblate/static/
<Directory /var/lib/weblate/static/>
    Require all granted
</Directory>

# DATA_DIR/media/
Alias /weblate/media/ /var/lib/weblate/media/
<Directory /var/lib/weblate/media/>
    Require all granted
</Directory>

WSGIScriptAlias /weblate /usr/share/weblate/weblate/wsgi.py/weblate
WSGIPassAuthorization On

<Directory /usr/share/weblate/weblate>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
</VirtualHost>

```

Additionally, you will have to adjust `weblate/settings.py`:

```
URL_PREFIX = '/weblate'
```

Background tasks using Celery

New in version 3.2.

Weblate uses Celery to process background tasks. The example settings come with eager configuration, which does process all tasks in place, but you want to change this to something more reasonable for a production setup.

A typical setup using Redis as a backend looks like this:

```

CELERY_TASK_ALWAYS_EAGER = False
CELERY_BROKER_URL = 'redis://localhost:6379'
CELERY_RESULT_BACKEND = CELERY_BROKER_URL

```

You should also start the Celery worker to process the tasks and start scheduled tasks, this can be done directly on the command line (which is mostly useful when debugging or developing):

```

./weblate/examples/celery start
./weblate/examples/celery stop

```

Most likely you will want to run Celery as a daemon and that is covered by [Daemonization](#). For the most common Linux setup using systemd, you can use the example files shipped in the `examples` folder listed below.

Systemd unit to be placed as `/etc/systemd/system/celery-weblate.service`:

```

[Unit]
Description=Celery Service (Weblate)
After=network.target

[Service]

```

(continues on next page)

(continued from previous page)

```

Type=forking
User=weblate
Group=weblate
EnvironmentFile=/etc/default/celery-weblate
WorkingDirectory=/home/weblate
PermissionsStartOnly=true
ExecStartPre=/bin/mkdir -p /var/run/celery
ExecStartPre=/bin/chown -R weblate /var/run/celery/
ExecStartPre=/bin/mkdir -p /var/log/celery
ExecStartPre=/bin/chown -R weblate /var/log/celery/
ExecStart=/bin/sh -c '${CELERY_BIN} multi start ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'
ExecStop=/bin/sh -c '${CELERY_BIN} multi stopwait ${CELERYD_NODES} \
  --pidfile=${CELERYD_PID_FILE}'
ExecReload=/bin/sh -c '${CELERY_BIN} multi restart ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'

```

[Install]

```
WantedBy=multi-user.target
```

Environment configuration to be placed as /etc/default/celery-weblate:

```

# Name of nodes to start
CELERYD_NODES="celery notify search memory"

# Absolute or relative path to the 'celery' command:
CELERY_BIN="/usr/local/bin/celery"

# App instance to use
# comment out this line if you don't use an app
CELERY_APP="weblate"

# Extra command-line arguments to the worker,
# increase concurrency if you get weblate.E019
CELERYD_OPTS="--beat:celery --concurrency:celery=4 --queues:celery=celery --prefetch-
multiplier:celery=4 \
--concurrency:notify=4 --queues:notify=notify --prefetch-multiplier:notify=4 \
--concurrency:search=1 --queues:search=search --prefetch-multiplier:search=2000 \
--concurrency:memory=1 --queues:memory=memory --prefetch-multiplier:memory=2000"

# Logging configuration
# - %n will be replaced with the first part of the nodename.
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race conditions.
CELERYD_PID_FILE="/var/run/celery/weblate-%n.pid"
CELERYD_LOG_FILE="/var/log/celery/weblate-%n%I.log"
CELERYD_LOG_LEVEL="INFO"

# Internal Weblate variable to indicate we're running inside Celery
CELERY_WORKER_RUNNING="1"

```

Logrotate configuration to be placed as /etc/logrotate.d/celery:

```

/var/log/celery/*.log {
    weekly

```

(continues on next page)

(continued from previous page)

```
missingok
rotate 12
compress
notifempty
}
```

Weblate comes with built-in setup for scheduled tasks. You can however define additional tasks in `settings.py`, for example see [Lazy commits](#).

You can use `celery_queues` to see current length of Celery task queues. In case the queue will get too long, you will also get configuration error in the admin interface.

Note: The Celery process has to be executed under the same user as Weblate and the WSGI process, otherwise files in the `DATA_DIR` will be stored with mixed ownership, leading to runtime issues.

Warning: The Celery errors are by default only logged into Celery log and are not visible to user. In case you want to have overview on such failures, it is recommended to configure [Collecting error reports](#).

See also:

Configuration and defaults, [Workers Guide](#), [Daemonization](#), [Monitoring and Management Guide](#), `celery_queues`

4.15.11 Monitoring Weblate

Weblate provides the `/healthz/` URL to be used in simple health checks, for example using Kubernetes.

4.15.12 Collecting error reports

Weblate, as any other software, can fail. In order to collect useful failure states we recommend to use third party services to collect such information. This is especially useful in case of failing Celery tasks, which would otherwise only report error to the logs and you won't get notified on them. Weblate has support for the following services:

Sentry

Weblate has built in support for [Sentry](#). To use it, it's enough to follow instructions for [Sentry for Python](#).

In short, you need to adjust `settings.py`:

```
import raven

# Add raven to apps:
INSTALLED_APPS = (
    # ... other app classes ...
    'raven.contrib.django.raven_compat',
)

RAVEN_CONFIG = {
    'dsn': 'https://id:key@your.sentry.example.com/',
    # Setting public_dsn will allow collecting user feedback on errors
```

(continues on next page)

(continued from previous page)

```
'public_dsn': 'https://id@your.sentry.example.com/',
# If you are using git, you can also automatically configure the
# release based on the git info.
'release': raven.fetch_git_sha(BASE_DIR),
}
```

Rollbar

Weblate has built-in support for [Rollbar](#). To use it, it's enough to follow instructions for [Rollbar notifier for Python](#).

In short, you need to adjust `settings.py`:

```
# Add rollbar as last middleware:
MIDDLEWARE = [
    # ... other middleware classes ...
    'rollbar.contrib.django.middleware.RollbarNotifierMiddleware',
]

# Configure client access
ROLLBAR = {
    'access_token': 'POST_SERVER_ITEM_ACCESS_TOKEN',
    'client_token': 'POST_CLIENT_ITEM_ACCESS_TOKEN',
    'environment': 'development' if DEBUG else 'production',
    'branch': 'master',
    'root': '/absolute/path/to/code/root',
}
```

Everything else is integrated automatically, you will now collect both server and client side errors.

4.15.13 Migrating Weblate to another server

Migrating Weblate to another server should be pretty easy, however it stores data in few locations which you should migrate carefully. The best approach is to stop Weblate for the migration.

Migrating database

Depending on your database backend, you might have several options to migrate the database. The most straightforward one is to dump the database on one server and import it on the new one. Alternatively you can use replication in case your database supports it.

The best approach is to use database native tools, as they are usually the most effective (e.g. **mysqldump** or **pg_dump**). If you want to migrate between different databases, the only option might be to use Django management to dump and import the database:

```
# Export current data
./manage.py dumpdata > /tmp/weblate.dump
# Import dump
./manage.py loaddata /tmp/weblate.dump
```

Migrating VCS repositories

The VCS repositories stored under `DATA_DIR` need to be migrated as well. You can simply copy them or use `rsync` to do the migration more effectively.

Migrating fulltext index

For the fulltext index, (stored in `DATA_DIR`) it is better not to migrate it, but rather generate a fresh one using `rebuild_index`.

Other notes

Don't forget to move other services Weblate might have been using like Redis, Cron jobs or custom authentication backends.

4.16 Weblate deployments

This is an overview of the supported deployment technologies.

4.16.1 Running Weblate with Docker

With dockerized Weblate deployment you can get your personal Weblate instance up and running in seconds. All of Weblate's dependencies are already included. PostgreSQL is set up as the default database.

Deployment

The following examples assume you have a working Docker environment, with `docker-compose` installed. Please check the Docker documentation for instructions.

1. Clone the weblate-docker repo:

```
git clone https://github.com/WeblateOrg/docker-compose.git weblate-docker
cd weblate-docker
```

2. Create a `docker-compose.override.yml` file with your settings. See [Docker environment variables](#) for full list of environment variables.

```
version: '3'
services:
  weblate:
    environment:
      WEBLATE_EMAIL_HOST: smtp.example.com
      WEBLATE_EMAIL_HOST_USER: user
      WEBLATE_EMAIL_HOST_PASSWORD: pass
      WEBLATE_SERVER_EMAIL: weblate@example.com
      WEBLATE_DEFAULT_FROM_EMAIL: weblate@example.com
      WEBLATE_ALLOWED_HOSTS: weblate.example.com,localhost
      WEBLATE_ADMIN_PASSWORD: password for the admin user
      WEBLATE_ADMIN_EMAIL: weblate.admin@example.com
```

Note: If `WEBLATE_ADMIN_PASSWORD` is not set, the admin user is created with a random password shown on first startup.

Append `'localhost'` to `WEBLATE_ALLOWED_HOSTS` to be able to access locally for testing.

You may also need to edit the `docker-compose.yml` file and change the default port from 80 if you already have a web server running on your local machine

3. Start Weblate containers:

```
docker-compose up
```

Enjoy your Weblate deployment, it's accessible on port 80 of the `weblate` container.

Changed in version 2.15-2: The setup has changed recently, priorly there was separate web server container, since 2.15-2 the web server is embedded in the Weblate container.

See also:

Invoking management commands

Docker container with HTTPS support

Please see [Deployment](#) for generic deployment instructions. To add a reverse HTTPS proxy an additional Docker container is required, `https-portal` will be used. This is made use of in the `docker-compose-https.yml` file. Then create a `docker-compose-https.override.yml` file with your settings:

```
version: '3'
services:
  weblate:
    environment:
      WEBLATE_EMAIL_HOST: smtp.example.com
      WEBLATE_EMAIL_HOST_USER: user
      WEBLATE_EMAIL_HOST_PASSWORD: pass
      WEBLATE_ALLOWED_HOSTS: weblate.example.com
      WEBLATE_ADMIN_PASSWORD: password for admin user
  https-portal:
    environment:
      DOMAINS: 'weblate.example.com -> http://weblate'
```

Whenever invoking `docker-compose` you need to pass both files to it, and then do:

```
docker-compose -f docker-compose-https.yml -f docker-compose-https.override.yml build
docker-compose -f docker-compose-https.yml -f docker-compose-https.override.yml up
```

Upgrading the Docker container

Usually it is good idea to only update the Weblate container and keep the PostgreSQL container at the version you have, as upgrading PostgreSQL is quite painful and in most cases does not bring many benefits.

You can do this by sticking with the existing `docker-compose` and just pull the latest images and then restart:

```
docker-compose stop
docker-compose pull
docker-compose up
```

The Weblate database should be automatically migrated on first startup, and there should be no need for additional manual actions.

Note: Upgrades across 3.0 are not supported by Weblate. If you are on 2.x series and want to upgrade to 3.x, first upgrade to the latest 3.0.1-x (at time of writing this it is the 3.0.1-7) image, which will do the migration and then continue upgrading to newer versions.

Docker environment variables

Many of Weblate's *Configuration* can be set in the Docker container using environment variables:

Generic settings

WEBLATE_DEBUG

Configures Django debug mode using *DEBUG*.

Example:

```
environment:
  WEBLATE_DEBUG: 1
```

See also:

Disable debug mode.

WEBLATE_LOGLEVEL

Configures the logging verbosity.

WEBLATE_SITE_TITLE

Configures the site-title shown on the heading of all pages.

WEBLATE_ADMIN_NAME

WEBLATE_ADMIN_EMAIL

Configures the site-admin's name and email.

Example:

```
environment:
  WEBLATE_ADMIN_NAME: Weblate admin
  WEBLATE_ADMIN_EMAIL: noreply@example.com
```

See also:

Properly configure admins

WEBLATE_ADMIN_PASSWORD

Sets the password for the admin user. If not set, the admin user is created with a random password shown on first startup.

Changed in version 2.9: Since version 2.9, the admin user is adjusted on every container startup to match *WEBLATE_ADMIN_PASSWORD*, *WEBLATE_ADMIN_NAME* and *WEBLATE_ADMIN_EMAIL*.

WEBLATE_SERVER_EMAIL

WEBLATE_DEFAULT_FROM_EMAIL

Configures the address for outgoing emails.

See also:

Configure email addresses

WEBLATE_ALLOWED_HOSTS

Configures allowed HTTP hostnames using [ALLOWED_HOSTS](#) and sets sitename to the first one.

Example:

```
environment:
  WEBLATE_ALLOWED_HOSTS: weblate.example.com,example.com
```

See also:

Allowed hosts setup, Set correct sitename

WEBLATE_SECRET_KEY

Configures the secret used by Django for cookie signing.

Deprecated since version 2.9: The secret is now generated automatically on first startup, there is no need to set it manually.

See also:

Django secret key

WEBLATE_REGISTRATION_OPEN

Configures whether registrations are open by toggling [REGISTRATION_OPEN](#).

Example:

```
environment:
  WEBLATE_REGISTRATION_OPEN: 0
```

WEBLATE_TIME_ZONE

Configures the used time zone in Weblate, see [TIME_ZONE](#).

Note: To change the time zone of the Docker container itself, use the TZ environment variable.

Example:

```
environment:
  WEBLATE_TIME_ZONE: Europe/Prague
```

WEBLATE_ENABLE_HTTPS

Makes Weblate assume it is operated behind a reverse HTTPS proxy, it makes Weblate use HTTPS in email and API links or set secure flags on cookies.

Note: This does not make the Weblate container accept HTTPS connections, you need to use a standalone reverse HTTPS proxy, see *Docker container with HTTPS support* for example.

Example:

```
environment:
  WEBLATE_ENABLE_HTTPS: 1
```

See also:

Set correct sitename

WEBLATE_IP_PROXY_HEADER

Lets Weblate fetching the IP address from any given HTTP header. Use this when using a reverse proxy in front of the Weblate container.

Enables *IP_BEHIND_REVERSE_PROXY* and sets *IP_PROXY_HEADER*.

Example:

```
environment:
  WEBLATE_IP_PROXY_HEADER: HTTP_X_FORWARDED_FOR
```

WEBLATE_REQUIRE_LOGIN

Configures login required for the whole of the Weblate installation using *LOGIN_REQUIRED_URLS*.

Example:

```
environment:
  WEBLATE_REQUIRE_LOGIN: 1
```

WEBLATE_LOGIN_REQUIRED_URLS_EXCEPTIONS

Adds URL exceptions for login required for the whole Weblate installation using *LOGIN_REQUIRED_URLS_EXCEPTIONS*.

WEBLATE_GOOGLE_ANALYTICS_ID

Configures ID for Google Analytics by changing *GOOGLE_ANALYTICS_ID*.

WEBLATE_GITHUB_USERNAME

Configures GitHub username for GitHub pull-requests by changing *GITHUB_USERNAME*.

See also:

Pushing changes to GitHub as pull request, Setting up hub

WEBLATE_SIMPLIFY_LANGUAGES

Configures the language simplification policy, see *SIMPLIFY_LANGUAGES*.

WEBLATE_AKISMET_API_KEY

Configures the Akismet API key, see *AKISMET_API_KEY*.

Machine translation settings

WEBLATE_MT_DEEPL_KEY

Enables *DeepL* machine translation and sets *MT_DEEPL_KEY*

WEBLATE_MT_GOOGLE_KEY

Enables *Google Translate* and sets *MT_GOOGLE_KEY*

WEBLATE_MT_MICROSOFT_COGNITIVE_KEY

Enables *Microsoft Cognitive Services Translator* and sets *MT_MICROSOFT_COGNITIVE_KEY*

WEBLATE_MT_MYMEMORY_ENABLED

Enables *MyMemory* machine translation and sets *MT_MYMEMORY_EMAIL* to *WEBLATE_ADMIN_EMAIL*.

WEBLATE_MT_GLOSBE_ENABLED

Enables *Glosbe* machine translation.

Authentication settings

LDAP

WEBLATE_AUTH_LDAP_SERVER_URI

WEBLATE_AUTH_LDAP_USER_DN_TEMPLATE

WEBLATE_AUTH_LDAP_USER_ATTR_MAP

WEBLATE_AUTH_LDAP_BIND_DN

WEBLATE_AUTH_LDAP_BIND_PASSWORD

LDAP authentication configuration.

Example:

```
environment:
  WEBLATE_AUTH_LDAP_SERVER_URI: ldap://ldap.example.org
  WEBLATE_AUTH_LDAP_USER_DN_TEMPLATE=uid=%(user)s,ou=People,dc=example,dc: net
  # map weblate 'full_name' to ldap 'name' and weblate 'email' attribute to 'mail'
  ↪ ' ldap attribute.
  # another example that can be used with OpenLDAP: 'full_name:cn,email:mail'
  WEBLATE_AUTH_LDAP_USER_ATTR_MAP: full_name:name,email:mail
```

See also:

LDAP authentication

GitHub

WEBLATE_SOCIAL_AUTH_GITHUB_KEY

WEBLATE_SOCIAL_AUTH_GITHUB_SECRET

Enables *GitHub authentication*.

BitBucket

WEBLATE_SOCIAL_AUTH_BITBUCKET_KEY

WEBLATE_SOCIAL_AUTH_BITBUCKET_SECRET

Enables *Bitbucket authentication*.

Facebook

WEBLATE_SOCIAL_AUTH_FACEBOOK_KEY

WEBLATE_SOCIAL_AUTH_FACEBOOK_SECRET

Enables *Facebook OAuth 2*.

Google

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET

Enables *Google OAuth 2*.

GitLab

WEBLATE_SOCIAL_AUTH_GITLAB_KEY

WEBLATE_SOCIAL_AUTH_GITLAB_SECRET

WEBLATE_SOCIAL_AUTH_GITLAB_API_URL

Enables *GitLab OAuth 2*.

Azure Active Directory

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_SECRET

Enables Azure Active Directory authentication, see [Microsoft Azure Active Directory](#).

Azure Active Directory with Tenant support

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID

Enables Azure Active Directory authentication with Tenant support, see [Microsoft Azure Active Directory](#).

Linux vendors

You can enable authentication using Linux vendors authentication services by setting following variables to any value.

WEBLATE_SOCIAL_AUTH_FEDORA

WEBLATE_SOCIAL_AUTH_OPENSUSE

WEBLATE_SOCIAL_AUTH_UBUNTU

Other authentication settings

WEBLATE_NO_EMAIL_AUTH

Disables email authentication when set to any value.

PostgreSQL database setup

The database is created by `docker-compose.yml`, so these settings affect both Weblate and PostgreSQL containers.

See also:

Database setup for Weblate

POSTGRES_PASSWORD

PostgreSQL password.

POSTGRES_USER

PostgreSQL username.

POSTGRES_DATABASE

PostgreSQL database name.

POSTGRES_HOST

PostgreSQL server hostname or IP address. Defaults to `database`.

POSTGRES_PORT

PostgreSQL server port. Defaults to none (uses the default value).

POSTGRES_SSL_MODE

Configure how PostgreSQL handles SSL in connection to the server, for possible choices see [SSL Mode Descriptions](#)

Caching server setup

Using Redis is strongly recommended by Weblate and you have to provide a Redis instance when running Weblate in Docker.

See also:

[Enable caching](#)

REDIS_HOST

The Redis server hostname or IP address. Defaults to `cache`.

REDIS_PORT

The Redis server port. Defaults to `6379`.

REDIS_DB

The Redis database number, defaults to `1`.

REDIS_PASSWORD

The Redis server password, not used by default.

Email server setup

To make outgoing email work, you need to provide a mail server.

See also:

[Configuring outgoing email](#)

WEBLATE_EMAIL_HOST

Mail server, the server has to listen on port 587 and understand TLS.

See also:

`EMAIL_HOST`

WEBLATE_EMAIL_PORT

Mail server port. Use if your cloud provider or ISP blocks outgoing connections on port 587.

See also:

`EMAIL_PORT`

WEBLATE_EMAIL_HOST_USER

Email authentication user, do NOT use quotes here.

See also:

`EMAIL_HOST_USER`

WEBLATE_EMAIL_HOST_PASSWORD

Email authentication password, do NOT use quotes here.

See also:

`EMAIL_HOST_PASSWORD`

WEBLATE_EMAIL_USE_SSL

Whether to use an implicit TLS (secure) connection when talking to the SMTP server. In most email documentation, this type of TLS connection is referred to as SSL. It is generally used on port 465. If you are experiencing problems, see the explicit TLS setting [WEBLATE_EMAIL_USE_TLS](#).

See also:

`EMAIL_USE_SSL`

WEBLATE_EMAIL_USE_TLS

Whether to use a TLS (secure) connection when talking to the SMTP server. This is used for explicit TLS connections, generally on port 587. If you are experiencing connections that hang, see the implicit TLS setting [WEBLATE_EMAIL_USE_SSL](#).

See also:

`EMAIL_USE_TLS`

Error reporting

It is recommended to collect errors from the installation in a systematic way, see [Collecting error reports](#).

To enable support for Rollbar, set the following:

ROLLBAR_KEY

Your Rollbar post server access token.

ROLLBAR_ENVIRONMENT

Your Rollbar environment, defaults to `production`.

To enable support for Sentry, set following:

SENTRY_DSN

Your Sentry DSN.

SENTRY_PUBLIC_DSN

Your Sentry public DSN.

SENTRY_ENVIRONMENT

Your Sentry environment, defaults to `production`.

Further configuration customization

You can additionally override the configuration in `/app/data/settings-override.py`. This is executed after all environment settings are loaded, so it gets completely set up, and can be used to customize anything.

Hub setup

In order to use the GitHub's pull-request feature, you must initialize hub configuration by entering the Weblate container and executing an arbitrary Hub command. For example:

```
docker-compose exec --user weblate weblate bash
cd
HOME=/app/data/home hub clone octocat/Spoon-Knife
```

The username passed for credentials must be the same as `GITHUB_USERNAME`.

See also:

Pushing changes to GitHub as pull request, Setting up hub

Select your machine - local or cloud providers

With docker-machine you can create your Weblate deployment either on your local machine, or on any large number of cloud-based deployments on e.g. Amazon AWS, Greenhost, and many other providers.

4.16.2 Running Weblate on OpenShift 2

This repository contains a configuration for the OpenShift platform as a service product, which facilitates easy installation of Weblate on OpenShift variants (see <https://www.openshift.com/> and <https://www.okd.io/>).

Prerequisites

1. OpenShift Account

You need an account on OpenShift Online (<https://www.openshift.com/>) or another OpenShift installation you have access to.

You can register a gratis account on OpenShift Online, which allows you to host up to 3 programs gratis.

2. OpenShift Client Tools

In order to follow the examples given in this documentation, you need to have the OpenShift Client Tools (RHC) installed: https://docs.openshift.com/online/cli_reference/get_started_cli.html

While there are other possibilities to create and configure OpenShift programs, this documentation is based on the OpenShift Client Tools (RHC) because they provide a consistent interface for all described operations.

Installation

You can install Weblate on OpenShift directly from Weblate's GitHub repository with the following command:

```
# Install Git from the development master branch
rhc -aweblate app create -t python-2.7 --from-code https://github.com/WeblateOrg/
↪weblate.git --no-git

# Install Weblate release
rhc -aweblate app create -t python-2.7 --from-code https://github.com/WeblateOrg/
↪weblate.git#weblate-3.7.1 --no-git
```

The `-a` option defines the name of your weblate installation, `weblate` in this instance. Feel free to specify a different name.

The above example installs the latest development version, you can optionally specify tag identifier to the right of the `#` sign to identify the version of Weblate to install. A list of available versions is available here: <https://github.com/WeblateOrg/weblate/tags>.

The `--no-git` option skips the creation of a local Git repository.

You can also specify which database you want to use:

```
# For MySQL
rhc -aweblate app create -t python-2.7 -t mysql-5.5 --from-code https://github.com/
↪WeblateOrg/weblate.git --no-git

# For PostgreSQL
rhc -aweblate app create -t python-2.7 -t postgresql-9.2 --from-code https://github.
↪com/WeblateOrg/weblate.git --no-git
```

Default Configuration

After installation on OpenShift, Weblate is ready for use and, preconfigured as follows:

- SQLite embedded database ([DATABASES](#))
- Random admin password
- Random Django secret key ([SECRET_KEY](#))
- Committing of pending changes if the Cron cartridge is installed ([commit_pending](#))
- Weblate machine translations for suggestions, based on previous translations ([MT_SERVICES](#))
- Weblate directories (STATIC_ROOT, [DATA_DIR](#), avatar cache) set according to OpenShift requirements/conventions.
- Django sitename and ALLOWED_HOSTS set to DNS name of your OpenShift program
- Email sender addresses set to no-reply@<OPENSHIFT_CLOUD_DOMAIN>, where <OPENSHIFT_CLOUD_DOMAIN> is the domain OpenShift runs under. In case of OpenShift Online it is rhcloud.com.

See also:

[Customize the Weblate Configuration](#)

Retrieve the Admin Password

Retrieve the generated admin password using the following command:

```
rhc -aweblate ssh credentials
```

Pending Changes

Weblate's OpenShift configuration contains a Cron job which periodically commits pending changes older than a certain age (24h by default). To enable the Cron job you need to add the Cron cartridge and restart Weblate as described in the previous section. You can change the age parameter by setting the environment variable `WEBLATE_PENDING_AGE` to the desired number of hours, e.g.:

```
rhc -aweblate env set WEBLATE_PENDING_AGE=48
```

Customize the Weblate Configuration

Customize the configuration of your Weblate installation on OpenShift through the use of environment variables. Override any of Weblate's settings documented under *Configuration* using `rhc env set` by prepending the settings name with `WEBLATE_`. The variable content is put into the configuration file verbatim, so it is parsed as a Python string, after replacing the environment variables in it (e.g. `$PATH`). To put in a literal `$` you need to escape it as `$$`.

For example override the `ADMINS` setting like this:

```
rhc -aweblate env set WEBLATE_ADMINS='(("John Doe", "john@example.org"),)'
```

To change the sitetitle, do not forget to include additional quotes:

```
rhc -aweblate env set WEBLATE_SITE_TITLE='"Custom Title"'
```

The new settings will only take effect once Weblate is restarted:

```
rhc -aweblate app stop
rhc -aweblate app start
```

Restarting using `rhc -aweblate app restart` does not work. For security reasons only constant expressions are allowed as values. With the exception of environment variables, which can be referenced using `${ENV_VAR}`. For example:

```
rhc -aweblate env set WEBLATE_SCRIPTS='("${OPENSIFT_DATA_DIR}/weblate/examples/hook-
↪unwrap-po",)'
```

You can check the effective settings Weblate is using by running:

```
rhc -aweblate ssh settings
```

This will also print syntax errors in your expressions. To reset a setting to its preconfigured value, just delete the corresponding environment variable:

```
rhc -aweblate env unset WEBLATE_ADMINS
```

See also:

Configuration

Updating

It is recommended that you try updates on a clone of your Weblate installation before running the actual update. To create such a clone, run:

```
rhc -aweblate2 app create --from-app weblate
```

Visit the newly given URL with a web browser and wait for the install/update page to disappear.

You can update your Weblate installation on OpenShift directly from Weblate's GitHub repository by executing:

```
rhc -aweblate2 ssh update https://github.com/WeblateOrg/weblate.git
```

The identifier to the right of the `#` sign identifies the version of Weblate to install. For a list of available versions see: <https://github.com/WeblateOrg/weblate/tags>. Please note that the update process will not work if you modified the Git repository of your Weblate installation. You can force an update by specifying the `--force` option with the update script. However any changes you made to the Git repository of your installation will be discarded:

```
rhc -aweblate2 ssh update --force https://github.com/WeblateOrg/weblate.git
```

The `--force` option is also needed when downgrading to an older version. Please note that only version 2.0 and newer can be installed on OpenShift, as older versions don't include the necessary configuration files.

The update script takes care of the following update steps, as described in *Generic upgrade instructions*.

- Install any new requirements
- `manage.py migrate`
- `manage.py setupgroups --move`
- `manage.py setuplang`
- `manage.py rebuild_index --all`
- `manage.py collectstatic --noinput`

4.16.3 Bitnami Weblate stack

Bitnami provides a Weblate stack for many platforms at <<https://bitnami.com/stack/weblate>>. The setup will be adjusted during installation, see <<https://bitnami.com/stack/weblate/README.txt>> for more documentation.

4.16.4 Weblate in YunoHost

The self-hosting project [YunoHost](#) provides a package for Weblate. Once you have your YunoHost installation, you may install Weblate as any other application. It will provide you with a fully working stack with backup and restoration, but you may still have to edit your settings file for specific usages.

You may use your administration interface, or this button (it will bring you to your server):



It also is possible to use the commandline interface:

```
yunohost app install https://github.com/YunoHost-Apps/weblate_ynh
```

4.17 Upgrading Weblate

4.17.1 Generic upgrade instructions

Before upgrading, please check the current *Software requirements* as they might have changed. Once all requirements are installed or updated, please adjust your `settings.py` to match changes in the configuration (consult `settings_example.py` for correct values).

Always check *Version specific instructions* before upgrade. In case you are skipping some versions, please follow instructions for all versions you are skipping in the upgrade. Sometimes it's better to upgrade to some intermediate version to ensure a smooth migration. Upgrading across multiple releases should work, but is not as well tested as single version upgrades.

Note: It is recommended to perform a full database backup prior to upgrade so that you can roll back the database in case upgrade fails, see [Backing up and moving Weblate](#).

1. Upgrade configuration file, refer to `settings_example.py` or [Version specific instructions](#) for needed steps.
2. Upgrade database structure:

```
./manage.py migrate --noinput
```

3. Collect updated static files (mostly javascript and CSS):

```
./manage.py collectstatic --noinput
```

4. Update language definitions (this is not necessary, but heavily recommended):

```
./manage.py setuplang
```

5. Optionally upgrade default set of privileges definitions (you might want to add new permissions manually if you have heavily tweaked access control):

```
./manage.py setupgroups
```

6. If you are running version from Git, you should also regenerate locale files every time you are upgrading. You can do this by invoking:

```
./manage.py compilemessages
```

7. Verify that your setup is sane (see also [Production setup](#)):

```
./manage.py check --deploy
```

8. Restart celery worker (see [Background tasks using Celery](#)).

4.17.2 Version specific instructions

Upgrade from 2.x

If you are upgrading from 2.x release, always first upgrade to 3.0.1 and then continue upgrading in the 3.x series. Upgrades skipping this step are not supported and will break.

See also:

[Upgrade from 2.20 to 3.0 in Weblate 3.0 documentation](#)

Upgrade from 3.0.1 to 3.1

Please follow [Generic upgrade instructions](#) in order to perform update.

Notable configuration or dependencies changes:

- Several no longer needed applications have been removed from `INSTALLED_APPS`.
- The settings now recommend using several Django security features, see [SSL/HTTPS](#).
- There is new dependency on the `jellyfish` module.

See also:

[Generic upgrade instructions](#)

Upgrade from 3.1 to 3.2

Please follow *[Generic upgrade instructions](#)* in order to perform update.

Notable configuration or dependencies changes:

- Rate limiting configuration has been changed, please see *[Rate limiting](#)*.
- Microsoft Terminology machine translation was moved to separate module and now requires `zeep` module.
- Weblate now uses Celery for several background tasks. There are new dependencies and settings because of this. You should also run Celery worker as standalone process. See *[Background tasks using Celery](#)* for more information.
- There are several changes in `settings_example.py`, most notable Celery configuration and middleware changes, please adjust your settings accordingly.

See also:

[Generic upgrade instructions](#)

Upgrade from 3.2 to 3.3

Please follow *[Generic upgrade instructions](#)* in order to perform update.

Notable configuration or dependencies changes:

- The `DEFAULT_CUSTOM_ACL` settings was replaced by `DEFAULT_ACCESS_CONTROL`. If you were using that please update your `settings.py`.
- Increase required `translate-toolkit` version to 2.3.1.
- Increase required social auth module versions (2.0.0 for `social-auth-core` and 3.0.0 for `social-auth-app-django`).
- The `CELERY_RESULT_BACKEND` should be now configured unless you are using eager mode, see *[Configuration and defaults](#)*.
- There is new `weblate.middleware.ProxyMiddleware` middleware needed if you use `IP_BEHIND_REVERSE_PROXY`.

See also:

[Generic upgrade instructions](#)

Upgrade from 3.3 to 3.4

Please follow *[Generic upgrade instructions](#)* in order to perform update.

Notable configuration or dependencies changes:

- The Celery now uses multiple queues, it is recommended to update to new worker setup which utilizes this, see *[Background tasks using Celery](#)*.
- There is new dependency on `diff-match-patch` and `translation-finder`.

See also:

[Generic upgrade instructions](#)

Upgrade from 3.4 to 3.5

Please follow *Generic upgrade instructions* in order to perform update.

Notable configuration or dependencies changes:

- There are several new checks included in the `CHECK_LIST`.

See also:

Generic upgrade instructions

Upgrade from 3.5 to 3.6

Please follow *Generic upgrade instructions* in order to perform update.

Notable configuration or dependencies changes:

- The automatic detection of file format has been removed. Please adjust your translation components configuration prior to upgrade. The upgrade should be able to gracefully handle most of situations, but can fail in some corner cases.
- If you have manually changed `WEBLATE_FORMATS`, you will have to remove `AutoFormat` from it.
- During the upgrade, the notifications settings need to be converted. This can be lengthy operation in case you have lot of users.

See also:

Generic upgrade instructions

Upgrade from 3.6 to 3.7

Please follow *Generic upgrade instructions* in order to perform update.

Notable configuration or dependencies changes:

- The Celery now uses separate queue for notifications, it is recommended to update to new worker setup which utilizes this, see *Background tasks using Celery*.
- There are new (bleach, gobject, pycairo) and updated (translation-finder) dependencies, you will now need Pango and Cairo system libraries as well, see *Pango and Cairo*.
- There are new addons, you might want to include them in case you modified the `WEBLATE_ADDONS`.
- There are new file formats, you might want to include them in case you modified the `WEBLATE_FORMATS`.
- There is change in the `CSRF_FAILURE_VIEW`.

See also:

Generic upgrade instructions

4.17.3 Upgrading from Python 2 to Python 3

Note: Weblate will support Python 2 until 4.0 release currently scheduled on April 2020. This is in line with Django dropping support for Python 2.

Weblate currently supports both Python 2.7 and 3.x. Upgrading existing installations is supported, but you should pay attention to some data stored on the disk as it might be incompatible between these two.

Things which might be problematic include Whoosh indices and file based caches. Fortunately these are easy to handle. Recommended upgrade steps:

1. Backup your *Translation Memory* using *dump_memory*:

```
./manage.py dump_memory > memory.json
```

2. Upgrade your installation to Python 3.

3. Delete *Translation Memory* database *delete_memory*:

```
./manage.py delete_memory --all
```

4. Restore your *Translation Memory* using *import_memory*.

```
./manage.py import_memory memory.json
```

5. Recreate fulltext index using *rebuild_index*:

```
./manage.py rebuild_index --clean --all
```

6. Cleanup avatar cache (if using file based) using *cleanup_avatar_cache*.

```
./manage.py cleanup_avatar_cache
```

7. It is recommended to throw away your caches.

4.17.4 Migrating from Pootle

As Weblate was originally written as replacement from Pootle, it is supported to migrate user accounts from Pootle. You can dump the users from Pootle and import them using *importusers*.

4.18 Backing up and moving Weblate

4.18.1 Backing up

Depending on what you want to save, back up the type data Weblate stores in each respective place.

Database

Where this is located depends on your database setup.

The database is the most important storage. Set up regular backups of your database, without it all your translation setup will be gone.

Native database backup

The recommended approach is to do dump of the database using database native tools such as **pg_dump** or **mysqldump**. It usually performs better than Django backup and restores complete tables with all data.

You can restore this backup in newer Weblate release, it will perform any necessary migrations when running in `migrate`. Please consult *Upgrading Weblate* on more detailed information how to perform upgrade between versions.

Django database backup

Alternatively you can backup database using Django's `dumpdata` command. That way the backup is database agnostic and can be used in case you want to change database backend.

Prior to restoring you need to be running exactly same Weblate version as was used when doing backups. This is necessary as the database structure does change between releases and you would end up corrupting the data in some way. After installing the same version, run all database migrations using `migrate`.

Once this is done, some entries will be already created in the database and you will have them in the database backup as well. The recommended approach is to delete such entries manually using management shell (see *Invoking management commands*):

```
./manage.py shell
>>> from weblate.auth.models import User
>>> User.objects.get(username='anonymous').delete()
```

Files

If you have enough backup space, simply backup the whole `DATA_DIR`. This is safe bet even if it includes some files you don't want. The following sections describe in detail what you should back up and what you can skip.

Dumped data for backups

Stored in `DATA_DIR/backups`.

Weblate dumps various data here, and you can include these files for more complete backups. The files are updated daily (requires a running Celery beats server, see *Background tasks using Celery*). Currently this includes:

- Translation memory dump, in JSON format.

Version control repositories

Stored in `DATA_DIR/vcs`.

The version control repositories contain a copy of your upstream repositories with Weblate changes. If you have push on commit enabled for all your translation components, all Weblate changes are included upstream and you do not have to backup the repositories on the Weblate side. They can be cloned again from the upstream locations with no data loss.

SSH and GPG keys

Stored in `DATA_DIR/ssh` and `DATA_DIR/home`.

If you are using SSH or GPG keys generated by Weblate, you should back up these locations, otherwise you will lose the private keys and you will have to regenerate new ones.

User uploaded files

Stored in `DATA_DIR/media`.

You should back up user uploaded files (e.g. *Visual context for strings*).

Translation memory

Stored in `DATA_DIR/memory`.

It is recommended to back up this content using `dump_memory` in JSON-, instead of binary format, as that might eventually change (and is also incompatible going from Python 2 to Python 3). Weblate prepares this dump daily, see *Dumped data for backups*.

Fulltext index

Stored in `DATA_DIR/whoosh`.

It is recommended to not backup this and regenerate it from scratch on restore.

4.18.2 Celery tasks

The Celery tasks queue might contain some info, but is usually not needed for a backup. At most you will lose updates that have not yet been processed to translation memory. It is recommended to perform the fulltext or repository updates upon restoring anyhow, so there is no problem in losing these.

See also:

Background tasks using Celery

4.18.3 Restoring

1. Restore all data you have backed up.
2. Recreate a fulltext index using `rebuild_index`:
3. Restore your *Translation Memory* using `import_memory`.

```
./manage.py rebuild_index --clean --all
```

```
./manage.py import_memory memory.json
```

4. Update all repositories using `updategit`.

```
./manage.py updategit --all
```

4.18.4 Moving a Weblate installation

Relocate your installation to a different system by following the backup and restore instructions above.

See also:

Upgrading from Python 2 to Python 3

4.19 Authentication

4.19.1 User registration

The default setup for Weblate is to use python-social-auth, a form on the website to handle registration of new users. After confirming their email a new user can contribute or authenticate by using one of the third party services.

You can also turn off registration of new users using `REGISTRATION_OPEN`.

The authentication attempts are subject to *Rate limiting*.

4.19.2 Authentication backends

The inbuilt solution of Django is used for authentication, including various social options to do so. Using it means you can import the user database of other Django based projects (see *Migrating from Pootle*).

Django can additionally be set up to authenticate against other means too.

4.19.3 Social authentication

Thanks to [Welcome to Python Social Auth's documentation!](#), Weblate support authentication using many third party services such as GitLab, Ubuntu, Fedora, etc.

Please check their documentation for generic configuration instructions in [Django Framework](#).

Note: By default, Weblate relies on third-party authentication services to provide a validated email address. If some of the services you want to use don't support this, please enforce email validation on the Weblate side by configuring `FORCE_EMAIL_VALIDATION` for them. For example:

```
SOCIAL_AUTH_OPENSUSE_FORCE_EMAIL_VALIDATION = True
```

See also:

[Pipeline](#)

Enabling individual backends is quite easy, it's just a matter of adding an entry to the `AUTHENTICATION_BACKENDS` setting and possibly adding keys needed for a given authentication method. Please note that some backends do not provide user email by default, you have to request it explicitly, otherwise Weblate will not be able to properly credit contributions users make.

See also:

[Python Social Auth backend](#)

OpenID authentication

For OpenID based services it's usually just a matter of enabling them. The following section enables OpenID authentication for OpenSUSE, Fedora and Ubuntu:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.email.EmailAuth',
    'social_core.backends.suse.OpenSUSEOpenId',
```

(continues on next page)

(continued from previous page)

```
'social_core.backends.ubuntu.UbuntuOpenId',
'social_core.backends.fedora.FedoraOpenId',
'weblate.accounts.auth.WeblateUserBackend',
)
```

See also:

[OpenId](#)

GitHub authentication

You need to register an application on GitHub and then tell Weblate all its secrets:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.github.GithubOAuth2',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_GITHUB_KEY = 'GitHub Client ID'
SOCIAL_AUTH_GITHUB_SECRET = 'GitHub Client Secret'
SOCIAL_AUTH_GITHUB_SCOPE = ['user:email']
```

The GitHub should be configured to have callback URL as `https://example.com/accounts/complete/github/`.

See also:

[GitHub](#)

Bitbucket authentication

You need to register an application on Bitbucket and then tell Weblate all its secrets:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.bitbucket.BitbucketOAuth',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_BITBUCKET_KEY = 'Bitbucket Client ID'
SOCIAL_AUTH_BITBUCKET_SECRET = 'Bitbucket Client Secret'
SOCIAL_AUTH_BITBUCKET_VERIFIED_EMAILS_ONLY = True
```

See also:

[Bitbucket](#)

Google OAuth 2

To use Google OAuth 2, you need to register an application on <<https://console.developers.google.com/>> and enable the Google+ API.

The redirect URL is `https://WEBLATE_SERVER/accounts/complete/google-oauth2/`

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.google.GoogleOAuth2',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = 'Client ID'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = 'Client secret'
```

See also:

[Google](#)

Facebook OAuth 2

As per usual with OAuth 2 services, you need to register your application with Facebook. Once this is done, you can set up Weblate to use it:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.facebook.FacebookOAuth2',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_FACEBOOK_KEY = 'key'
SOCIAL_AUTH_FACEBOOK_SECRET = 'secret'
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email', 'public_profile']
```

See also:

[Facebook](#)

GitLab OAuth 2

For using GitLab OAuth 2, you need to register an application on <<https://gitlab.com/profile/applications>>.

The redirect URL is `https://WEBLATE_SERVER/accounts/complete/gitlab/` and ensure you mark the `read_user` scope.

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.gitlab.GitLabOAuth2',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)
```

(continues on next page)

(continued from previous page)

```
# Social auth backends setup
SOCIAL_AUTH_GITLAB_KEY = 'Application ID'
SOCIAL_AUTH_GITLAB_SECRET = 'Secret'
SOCIAL_AUTH_GITLAB_SCOPE = ['read_user']

# If you are using your own GitLab
# SOCIAL_AUTH_GITLAB_API_URL = 'https://gitlab.example.com/'
```

See also:

GitLab

Turning off password authentication

Email and password authentication can be disabled by removing `social_core.backends.email.EmailAuth` from `AUTHENTICATION_BACKENDS`. Always keep `weblate.accounts.auth.WeblateUserBackend` there, it is needed for core Weblate functionality.

Tip: You can still use password authentication for the admin interface, for users you manually create there. Just navigate to `/admin/`.

For example authentication using only the openSUSE Open ID provider can be achieved using the following:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.suse.OpenSUSEOpenId',
    'weblate.accounts.auth.WeblateUserBackend',
)
```

4.19.4 Password authentication

The default `settings.py` comes with a reasonable set of `AUTH_PASSWORD_VALIDATORS`:

- Passwords can't be too similar to your other personal info.
- Passwords must contain at least 6 characters.
- Passwords can't be a commonly used password.
- Passwords can't be entirely numeric.
- Passwords can't consist of a single character or only whitespace.
- Passwords can't match a password you have used in the past.

You can customize this setting to match your password policy.

Additionally you can also install `django-zxcvbn-password` which gives quite realistic estimates of password difficulty and allows rejecting passwords below a certain threshold.

4.19.5 LDAP authentication

LDAP authentication can be best achieved using the `django-auth-ldap` package. You can install it via usual means:


```
# Using PyPI
pip install django-auth-ldap>=1.3.0

# Using apt-get
apt-get install python-django-auth-ldap
```

Warning: With django-auth-ldap older than 1.3.0 the *Automatic group assignments* will not work properly for newly created users.

Note: There are some incompatibilities in the Python LDAP 3.1.0 module, which might prevent you from using that version. If you get error `AttributeError: 'module' object has no attribute '_trace_level'`, downgrading python-ldap to 3.0.0 might help.

Once you have the package installed, you can hook it into the Django authentication:

```
# Add LDAP backed, keep Django one if you want to be able to login
# even without LDAP for admin account
AUTHENTICATION_BACKENDS = (
    'django_auth_ldap.backend.LDAPBackend',
    'weblate.accounts.auth.WeblateUserBackend',
)

# LDAP server address
AUTH_LDAP_SERVER_URI = 'ldaps://ldap.example.net'

# DN to use for authentication
AUTH_LDAP_USER_DN_TEMPLATE = 'cn=%(user)s,o=Example'
# Depending on your LDAP server, you might use a different DN
# like:
# AUTH_LDAP_USER_DN_TEMPLATE = 'ou=users,dc=example,dc=com'

# List of attributes to import from LDAP upon login
# Weblate stores full name of the user in the full_name attribute
AUTH_LDAP_USER_ATTR_MAP = {
    'full_name': 'name',
    # Use the following if your LDAP server does not have full name
    # Weblate will merge them later
    # 'first_name': 'givenName',
    # 'last_name': 'sn',
    # Email is required for Weblate (used in VCS commits)
    'email': 'mail',
}
```

If you can not use direct bind for authentication, you will need to use search, and provide a user to bind for the search. For example:

```
import ldap
from django_auth_ldap.config import LDAPSearch

AUTH_LDAP_BIND_DN = ""
AUTH_LDAP_BIND_PASSWORD = ""
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
```

Note: You should remove 'social_core.backends.email.EmailAuth' from the AUTHENTICATION_BACKENDS setting, otherwise users will be able to set their password in Weblate, and authenticate using that. Keeping 'weblate.accounts.auth.WeblateUserBackend' is still needed in order to make permissions and facilitate anonymous users. It will also allow you to log in using a local admin account, if you have created it (e.g. by using `createadmin`).

See also:

[Django Authentication Using LDAP, Authentication](#)

4.19.6 CAS authentication

CAS authentication can be achieved using a package such as *django-cas-ng*.

Step one is disclosing the email field of the user via CAS. This has to be configured on the CAS server itself, and requires you run at least CAS v2 since CAS v1 doesn't support attributes at all.

Step two is updating Weblate to use your CAS server and attributes.

To install *django-cas-ng*:

```
pip install django-cas-ng
```

Once you have the package installed you can hook it up to the Django authentication system by modifying the settings.py file:

```
# Add CAS backed, keep the Django one if you want to be able to log in
# even without LDAP for the admin account
AUTHENTICATION_BACKENDS = (
    'django_cas_ng.backends.CASBackend',
    'weblate.accounts.auth.WeblateUserBackend',
)

# CAS server address
CAS_SERVER_URL = 'https://cas.example.net/cas/'

# Add django_cas_ng somewhere in the list of INSTALLED_APPS
INSTALLED_APPS = (
    ...,
    'django_cas_ng'
)
```

Finally, a signal can be used to map the email field to the user object. For this to work you have to import the signal from the *django-cas-ng* package and connect your code with this signal. Doing this in settings file can cause problems, therefore it's suggested to put it:

- In your app config's `django.apps.AppConfig.ready()` method (Django 1.7 and above)
- At the end of your `models.py` file (Django 1.6 and below)
- In the project's `urls.py` file (when no models exist)

```
from django_cas_ng.signals import cas_user_authenticated
from django.dispatch import receiver
@receiver(cas_user_authenticated)
def update_user_email_address(sender, user=None, attributes=None, **kwargs):
    # If your CAS server does not always include the email attribute
```

(continues on next page)

(continued from previous page)

```
# you can wrap the next two lines of code in a try/catch block.
user.email = attributes['email']
user.save()
```

See also:[Django CAS NG](#)

4.19.7 Configuring third party Django authentication

Generally any Django authentication plugin should work with Weblate. Just follow the instructions for the plugin, just remember to keep the Weblate user backend installed.

See also:[LDAP authentication](#), [CAS authentication](#)

Typically the installation will consist of adding an authentication backend to `AUTHENTICATION_BACKENDS` and installing an authentication app (if there is any) into `INSTALLED_APPS`:

```
AUTHENTICATION_BACKENDS = (
    # Add authentication backend here
    'weblate.accounts.auth.WeblateUserBackend',
)

INSTALLED_APPS = (
    ...
    'weblate',
    # Install authentication app here
)
```

4.20 Access control

Changed in version 3.0: Before Weblate 3.0, the privilege system was based on Django, but is now specifically built for Weblate. If you are using an older version, please consult documentation for that version, the information here will not apply.

Weblate comes with a fine grained privilege system to assign user permissions for the whole instance, or in a limited scope.

The permission system based on groups and roles, where roles define a set of permissions, and groups assign them to users and translations, see [Users, roles, groups and permissions](#) for more details.

After installation a default set of groups is created, and you can use those to assign users roles for the whole instance (see [Default groups and roles](#)). Additionally when [Per project access control](#) is turned on, you can assign users to specific translation projects. More fine-grained configuration can be achieved using [Custom access control](#)

4.20.1 Common setups

Locking down Weblate

To completely lock down your Weblate installation, you can use `LOGIN_REQUIRED_URLS` to force users to log in and `REGISTRATION_OPEN` to prevent new registrations.

Site wide permissions

To manage permissions for a whole instance, just add users to *Users* (this is done by default using the *Automatic group assignments*), *Reviewers* and *Managers* groups. Keep all projects configured as *Public* (see *Per project access control*).

Per project permissions

Set your projects to *Protected* or *Private*, and manage users per project in the Weblate interface.

Adding permissions to languages, projects or component sets

You can additionally grant permissions to any user based on project, language or a component set. To achieve this, create a new group (e.g. *Czech translators*) and configure it for a given resource. Any assigned permissions will be granted to members of that group for selected resources.

This will work just fine without additional setup, if using per project permissions. For permissions on the whole instance, you will probably also want to remove these permissions from the *Users* group, or change automatic assignment of all users to that group (see *Automatic group assignments*).

4.20.2 Per project access control

Note: By enabling ACL, all users are prohibited from accessing anything within a given project, unless you add the permissions for them to do just that.

You can limit user's access to individual projects. This feature is turned on by *Access control* in the configuration of each respective project. This automatically creates several groups for this project, see *Predefined groups*.

The following choices exist for *Access control*:

Public Publicly visible and translatable

Protected Publicly visible, but translatable only for selected users

Private Visible and translatable only for selected users

Custom Weblate does not manage users, see *Custom access control*.

Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Manage users

Users

Username	Full name	Email	Administration	Billing	Glossary	Languages	Memory	Screenshots	Template	Translate	VCS	
testuser	Weblate Test	weblate@example.org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Remove

The user will be removed from the project once all user permissions are removed.

Add new user

User to add

Please provide username or email. User needs to already have an active account in Weblate.

Add

Project access control

Access control

Protected

How to restrict access to this project is detailed in the documentation.

Public
Publicly visible and translatable

Protected
Publicly visible, only translatable for chosen users

Private
Visible and translatable only for chosen users

Custom
Only use this if you know what you are doing, enabling it might revoke your access to this project. Permissions are not managed in Weblate.

☐ Enable reviews
Requires dedicated reviewers to approve translations.

You do not have permission to change project access control.

Check your billing status

Powered by Weblate 3.7.1
About Weblate
Legal
Contact
Documentation
Donate to Weblate

To allow access to this project, you have to add the privilege either directly to the given user, or group of users in the Django admin interface, or by using user management on the project page, as described in *Managing per project access control*.

Note: Even with ACL turned on, some summary info will be available about your project:

- Statistics for the whole instance, including counts for all projects.
- Language summary for the whole instance, including counts for all projects.

4.20.3 Automatic group assignments

You can set up Weblate to automatically add users to groups based on their email addresses. This automatic assignment happens only at the time of account creation.

This can be set up in the Django admin interface (in the *Accounts* section).

Note: The automatic group assignment for the *Users* and *Viewers* groups will always be created by Weblate upon migrations, in case you want to turn it off, simply set the regular expression to `^$`, which will never match.

4.20.4 Users, roles, groups and permissions

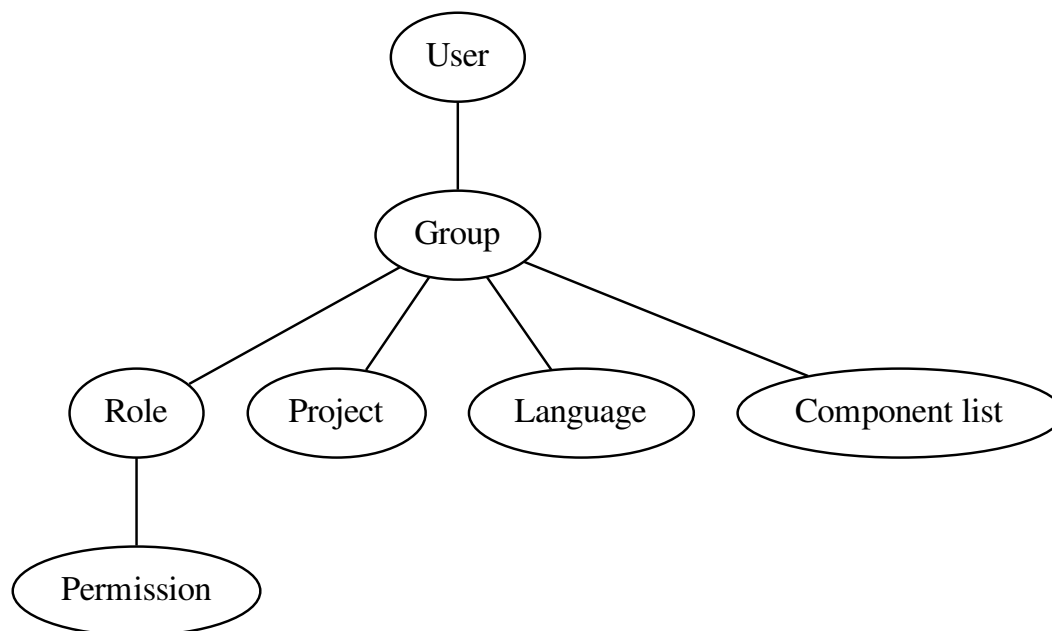
The authentication models consist of several objects:

Permission Individual permissions defined by Weblate. You can not assign individual permissions, this can only be done through assignment of roles.

Role Role defines a set of permissions. This allows reuse of these sets in several places, and makes the administration easier.

User Users can be members of several groups.

Group Groups connect roles, users and authentication objects (projects, languages and component lists).



Permission checking

Whenever a permission is checked to decide whether one is able to perform a given action, the check is carried out according to scope, and the following checks are performed:

Project Compared against the scope of the project, if not set, this matches no project.

You can use *Project selection* to automate inclusion of all projects.

Component list The scope component is matched against this list, if not set, this is ignored.

Obviously this has no effect when checking access of the project scope, so you will have to grant access to view all projects in a component list by other means. By default this is achieved through the use of the *Viewers* group, see *Default groups and roles*).

Language Compared against scope of translations, if not set, this matches no language.

You can use *Language selection* to automate inclusion of all languages.

Checking access to a project

A user has to be a member of a group linked to the project. Only membership is enough, no specific permissions are needed to access a project (this is used in the default *Viewers* group, see *Default groups and roles*).

4.20.5 Managing users and groups

All users and groups can be managed using the Django admin interface, available under `/admin/` URL.

Managing per project access control

Note: This feature only works for ACL controlled projects, see *Per project access control*.

Users with the *Can manage ACL rules for a project* privilege (see *Access control*) can also manage users in projects with access control turned on through the project page. You can add users, or remove them from a project, or make them owners of it.

The user management is available in the *Tools* menu of a project:

Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Manage users

Users

Username	Full name	Email	Administration	Billing	Glossary	Languages	Memory	Screenshots	Template	Translate	VCS	
testuser	Weblate Test	weblate@example.org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Remove

The user will be removed from the project once all user permissions are removed.

Add new user

User to add

Please provide username or email. User needs to already have an active account in Weblate.

Add

Project access control

Access control

Protected

How to restrict access to this project is detailed in the documentation.

Public
Publicly visible and translatable

Protected
Publicly visible, only translatable for chosen users

Private
Visible and translatable only for chosen users

Custom
Only use this if you know what you are doing, enabling it might revoke your access to this project. Permissions are not managed in Weblate.

☐ Enable reviews
Requires dedicated reviewers to approve translations.

You do not have permission to change project access control.

Check your billing status

Powered by Weblate 3.7.1

About Weblate
Legal
Contact
Documentation
Donate to Weblate

See also:

Per project access control

Predefined groups

Weblate comes with a predefined set of groups for a project, wherefrom you can assign users.

Administration

Has all permissions available in the project.

Glossary

Can manage glossary (add or remove entries, or upload).

Languages

Can manage translated languages - add or remove translations.

Screenshots

Can manage screenshots - add or remove them, and associate them to source strings.

Template

Can edit translation templates in *Monolingual components* and source string info.

Translate

Can translate the project, and upload translations made offline.

VCS

Can manage VCS and access the exported repository.

Review

Can approve translations during review.

Billing

Can access billing info (see *Billing*).

4.20.6 Custom access control

By choosing *Custom* as *Access control*, Weblate will stop managing access for a given project, and you can set up custom rules in the Django admin interface. This can be used to define more complex access control, or set up a shared access policy for all projects in a single Weblate instance. If you want to enable this for all projects by default, please configure the *DEFAULT_ACCESS_CONTROL*.

Warning: By turning this on, Weblate will remove all *Per project access control* it has created for this project. If you are doing this without admin permission from the instance, you will instantly loose your access to manage the project.

4.20.7 Default groups and roles

List of privileges

Billing (see *Billing*) View billing info [*Administration*, *Billing*]

Changes Download changes [*Administration*]

Comments Post comment [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*] Delete comment [*Administration*]

Component Edit component settings [*Administration*] Lock component, preventing it from being translated [*Administration*]

Glossary Add glossary entry [*Administration*, *Manage glossary*, *Power user*] Edit glossary entry [*Administration*, *Manage glossary*, *Power user*] Delete glossary entry [*Administration*, *Manage glossary*, *Power user*] Upload glossary entries [*Administration*, *Manage glossary*, *Power user*]

Machinery Use machine translation services [*Administration*, *Power user*]

Projects Edit project settings [*Administration*] Manage project access [*Administration*]

Reports Download reports [*Administration*]

Screenshots Add screenshot [*Administration*, *Manage screenshots*] Edit screenshot [*Administration*, *Manage screenshots*] Delete screenshot [*Administration*, *Manage screenshots*]

Source strings Edit source string info [*Administration*, *Edit source*]

Strings Add new strings [*Administration*] Ignore failing checks [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*] Edit strings [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*] Review

strings [*Administration, Review strings*] Edit string when suggestions are enforced [*Administration, Review strings*] Edit source strings [*Administration, Edit source, Power user*]

Suggestions Accept suggestions [*Administration, Edit source, Power user, Review strings, Translate*] Add suggestions [*Add suggestion, Administration, Edit source, Power user, Review strings, Translate*] Delete suggestions [*Administration*] Vote on suggestions [*Administration, Edit source, Power user, Review strings, Translate*]

Translations Start new translation [*Administration, Manage languages, Power user*] Perform automatic translation [*Administration, Manage languages*] Delete existing translations [*Administration, Manage languages*] Start translation into a new language [*Administration, Manage languages*]

Uploads Define author of translation upload [*Administration*] Overwrite existing strings with an upload [*Administration, Edit source, Power user, Review strings, Translate*] Upload translation strings [*Administration, Edit source, Power user, Review strings, Translate*]

VCS Access the internal repository [*Access repository, Administration, Manage repository, Power user*] Commit changes to the internal repository [*Administration, Manage repository*] Push change from the internal repository [*Administration, Manage repository*] Reset changes in the internal repository [*Administration, Manage repository*] View upstream repository location [*Access repository, Administration, Manage repository, Power user*] Update the internal repository [*Administration, Manage repository*]

Global privileges Use management interface (global) Add language definitions (global) Manage language definitions (global) Add groups (global) Manage groups (global) Add users (global) Manage users (global) Manage whiteboard (global) Manage translation memory (global)

Note: The global privileges are not granted to any default role. These are powerful and they are quite close to the superuser status - most of them can affect all projects on your Weblate installation.

List of groups

The following groups are created upon installation (or after executing *setupgroups*):

Guests Defines permissions for non authenticated users.

This group contains only anonymous users (see *ANONYMOUS_USER_NAME*).

You can remove roles from this group to limit permissions for non authenticated users.

Default roles: *Add suggestion, Access repository*

Viewers This role ensures visibility of public projects for all users. By default all users are members of this group.

By default all users are members of this group, using *Automatic group assignments*.

Default roles: none

Users Default group for all users.

By default all users are members of this group using *Automatic group assignments*.

Default roles: *Power user*

Reviewers Group for reviewers (see *Translation workflows*).

Default roles: *Review strings*

Managers Group for administrators.

Default roles: *Administration*

Warning: Never remove the predefined Weblate groups and users, this can lead to unexpected problems. If you do not want to use these features, just remove all privileges from them.

4.21 Translation projects

4.21.1 Translation organization

Weblate organizes translatable content into a tree-like structure. The bottom level object is *Project configuration*, which should hold all translations belonging together (for example translation of an application in several versions and/or accompanying documentation). On the level above, *Component configuration*, which is actually the component to translate. Here you define the VCS repository to use, and the mask of files to translate. Above *Component configuration* there are individual translations, handled automatically by Weblate as translation files (which match the mask defined in *Component configuration*) appear in the VCS repository.

All translation components need to be available as VCS repositories, and are organized in a project/component structure.

Weblate supports a wide range of translation formats (both bilingual and monolingual ones) supported by Translate Toolkit, see *Supported file formats* for more info.

Note: You can share cloned VCS repositories using *Weblate internal URLs*. Using this feature is highly recommended when you have many components sharing the same VCS. It improves performance and decreases the required disk space.

4.21.2 Adding translation projects and components

Changed in version 3.2: Since the 3.2 release the interface for adding projects and components is included in Weblate, and no longer requires you to use *Django admin interface*.

Changed in version 3.4: As of 3.4, the process of adding components is multi staged, with automated discovery of most parameters.

Based on your permissions, you can create new translation projects and components in Weblate. It is always permitted for superusers, and if your instance uses billing (e.g. like <https://hosted.weblate.org/> see *Billing*), you can also create those based on your plans allowance.

You can view your current billing plan on a separate page:

[Dashboard](#)
[Projects](#)
[Your languages](#)
[Languages](#)

[+ -](#)
[?](#)

Your profile / Billing

Billing plan

Current plan	Basic plan (Active)	See other plans
Monthly price	19 EUR	
Yearly price	199 EUR	
Strings limit	Used 0	<div></div>
Languages limit	Used 0	<div></div>
Repositories limit	Used 0	<div></div>
Last invoice	2019-06-27 - 2019-06-29	
Projects limit	Used 0 of 1	<div></div>
Projects	<div>No projects currently assigned!</div> <div>Add new translation project</div>	
<div>Terminate billing plan</div>		

Invoices

Invoice period	Invoice amount	Download invoice
06/27/2019 - 06/29/2019	19.0 EUR	Not available

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

The project creation can be initiated from there, or using the menu in the navigation bar, filling in basic info about the translation project to complete addition of it:

[Dashboard](#)
[Projects](#)
[Your languages](#)
[Languages](#)

[+ -](#)
[?](#)

Create project

Add new translation project

Project name

Display name

URL slug

Name used in URLs and filenames.

Project website

Main website of translated project.

Mailing list

Mailing list for translators.

Translation instructions

URL with instructions for translators.

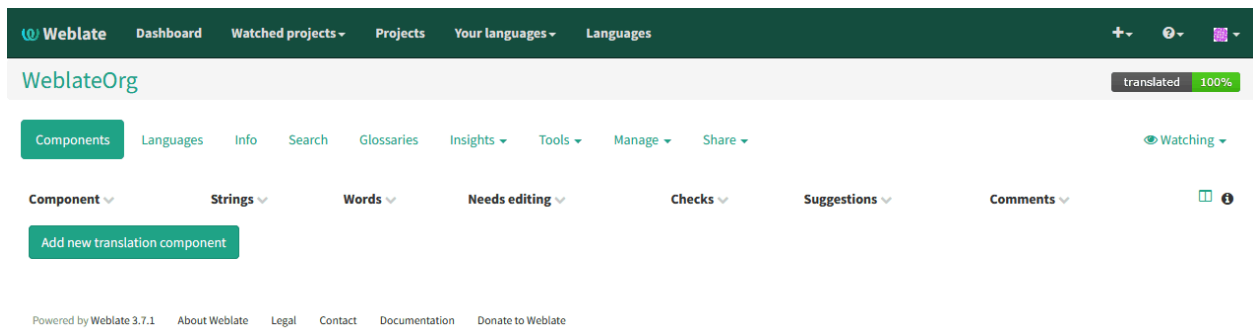
Billing

Weblate Test (Basic plan)

Save

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

After creating the project, you are taken directly to the project page:




Creating a new translation component can be initiated via a single click there. The process of creating a component is multi-staged and automatically detects most translation parameters.

First you need to fill in name and repository location:

The screenshot displays the 'Create component' form in Weblate. The form is titled 'Add new translation component' and contains several input fields and dropdown menus. The 'Component name' field is filled with 'Language names'. The 'URL slug' field is filled with 'language-names'. The 'Project' dropdown menu is set to 'WeblateOrg'. The 'Version control system' dropdown menu is set to 'Git'. The 'Source code repository' field is filled with 'https://github.com/WeblateOrg/demo.git'. The 'Repository branch' field is empty. A green 'Continue' button is located at the bottom of the form. The footer of the page is identical to the one in the previous screenshot.

On the next page, you are presented with a list of discovered translatable resources:

 Weblate

Dashboard

Watched projects ▾

Projects

Your languages ▾

Languages

+ ▾

🔍 ▾

🌐 ▾

Create component

Add new translation component

Choose translation files to import

☐ Specify configuration manually

☒ File format `Android String Resource`, Filemask `app/src/main/res/values-*/strings.xml`

☐ File format `Gettext PO file`, Filemask `weblate/langdata/locale/*/LC_MESSAGES/django.po`

☐ File format `Gettext PO file`, Filemask `weblate/locale/*/LC_MESSAGES/django.po`

☐ File format `Gettext PO file`, Filemask `weblate/locale/*/LC_MESSAGES/djangojs.po`

Continue

Powered by Weblate 3.7.1

[About Weblate](#)

[Legal](#)

[Contact](#)

[Documentation](#)

[Donate to Weblate](#)

As a last step, you review the translation component info and fill in optional details:

See also:

Django admin interface, *Project configuration*, *Component configuration*

4.21.3 Project configuration

To add a new component for translation, you need to create a translation project first. The project is like a shelf, in which real translations are stacked. All components in the same project share suggestions and their dictionary; the translations are also automatically propagated through all components in a single project (unless turned off in the component configuration).

The project has only a few attributes that informs translators of it:

Project website URL where translators can find more info about the project.

Mailing list Mailing list where translators can discuss or comment translations.

Translation instructions URL to more site with more detailed instructions for translators.

Set Language-Team header Whether Weblate should manage the Language-Team header (this is a *GNU Gettext* only feature right now).

Use shared translation memory Whether to use shared translation memory, see *Shared translation memory* for more details.

Access control Configure per project access control, see *Per project access control* for more details.

Enable reviews Enable review workflow, see *Dedicated reviewers*.

Enable hooks Whether unauthenticated *Notification hooks* are to be used for this repository.

Source language Language used for source strings in all components. Change this if you are translating from something else than English.

Note: Most of the fields can be edited by project owners or managers, in the Weblate interface.

Adjusting interaction

There are also additional features which you can control, like automatic pushing of changes (see also *Pushing changes*) or maintainership of the Language-Team header.

4.21.4 Component configuration

A component is a grouping of something for translation. You enter a VCS repository location and file mask for which files you want translated, and Weblate automatically fetches from this VCS, and finds all matching translatable files.

You can find some examples of typical configurations in the *Supported file formats*.

Note: It is recommended to keep translation components to a reasonable size - split the translation by anything that makes sense in your case (individual apps or addons, book chapters or websites).

Weblate easily handles translations with 10000s of strings, but it is harder to split work and coordinate among translators with such large translation components.

Should the language definition for a translation be missing, an empty definition is created and named as “cs_CZ (generated)”. You should adjust the definition and report this back to the Weblate authors, so that the missing languages can be included in next release.

The component contains all important parameters for working with the VCS, and for getting translations out of it:

Version control system VCS to use, see [Version control integration](#) for details.

Source code repository VCS repository used to pull changes, see [Accessing repositories](#) for more details.

This can either be a real VCS URL or `weblate://project/component` indicating that the repository should be shared with another component. See [Weblate internal URLs](#) for more details.

Repository push URL Repository URL used for pushing. This is completely optional and push support is turned off when this is empty. See [Accessing repositories](#) for more details on how to specify a repository URL.

Repository browser URL of repository browser used to display source files (location of used messages). When empty, no such links will be generated. You can use [Template markup](#).

For example on GitHub, use something like `https://github.com/WeblateOrg/hello/blob/{{branch}}/{{filename}}#L{{line}}`.

Exported repository URL URL where changes made by Weblate are exported. This is important when [Continuous localization](#) is not used, or when there is a need to manually merge changes. You can use [Git exporter](#) to automate this for Git repositories.

Repository branch Which branch to checkout from the VCS, and where to look for translations.

File mask Mask of files to translate, including path. It should include one “*” replacing language code (see [Language definitions](#) for info on how this is processed). In case your repository contains more than one translation file (e.g. more Gettext domains), you need to create a component for each of them.

For example `po/*.po` or `locale/*/LC_MESSAGES/django.po`.

In case your filename contains special chars such as `[,]`, these need to be escaped as `[[]]` or `[]]`.

Monolingual base language file Base file containing string definitions for [Monolingual components](#).

Edit base file Whether to allow editing the base file for [Monolingual components](#).

Base file for new translations Base file used to generate new translations, e.g. `.pot` file with Gettext, see [Adding new translations](#) for more info.

File format Translation file format, see also [Supported file formats](#).

Source string bug report address Email address used for reporting upstream bugs. This address will also receive notification about any source string comments made in Weblate.

Locked You can lock the translation to prevent updates by users.

Allow translation propagation You can turn off propagation of translations to this component from other components within same project. This really depends on what you are translating, sometimes it’s desirable to have make use of a translation more than once.

It’s usually a good idea to turn this off for monolingual translations, unless you are using the same IDs across the whole project.

Save translation history Whether to store a history of translation changes in the database.

Enable suggestions Whether translation suggestions are accepted for this component.

Suggestion voting Turns onotecasting for suggestions, see [Suggestion voting](#).

Autoaccept suggestions Automatically accept voted suggestions, see [Suggestion voting](#).

Translation flags Customization of quality checks and other Weblate behavior, see [Customizing behavior](#).

Translation license License of the translation, (does not need to be the same as the source code license).

License URL URL where users can find the actual text of a license in full.

New translation How to handle requests for creation of new languages. See *Adding new translations*.

Language code style Customize language code used to generate the filename for translations created by Weblate, see *Adding new translations* for more details.

Merge style You can configure how updates from the upstream repository are handled. This might not be supported for some VCSs. See *Merge or rebase* for more details.

Commit message Message used when committing a translation, see *Template markup*.

Committer name Name of the committer used for Weblate commits, the author will always be the real translator. On some VCSs this might be not supported. Default value can be changed by *DEFAULT_COMMITTER_NAME*.

Committer email Email of committer used for Weblate commits, the author will always be the real translator. On some VCSs this might be not supported. Default value can be changed by *DEFAULT_COMMITTER_EMAIL*.

Push on commit Whether committed changes should be automatically pushed to the upstream repository.

Age of changes to commit Sets how old changes (in hours) are to get before they are committed by *commit_pending* management command (usually executed by Cron). The Default value can be changed by *COMMIT_PENDING_HOURS*.

Language filter Regular expression used to filter the translation when scanning for file mask. This can be used to limit the list of languages managed by Weblate (e.g. `^(cs|de|es)$` will include only these languages. Please note that you need to list language codes as they appear in the filename.

Note: Most of the fields can be edited by project owners or managers, in the Weblate interface.

See also:

Does Weblate support other VCS than Git and Mercurial?, Translation component alerts

4.21.5 Template markup

Weblate uses simple markup language in several places where text rendering is needed. It is based on *The Django template language*, so it can be quite powerful.

Currently it is used in:

- Commit message formatting, see *Component configuration*
- **Several addons**
 - *Component discovery*
 - *Statistics generator*
 - *Executing scripts from addon*

There following variables are available in the component templates:

```
{{ language_code }} Language code
{{ language_name }} Language name
{{ component_name }} Component name
{{ component_slug }} Component slug
{{ project_name }} Project name
```

`{{ project_slug }}` Project slug
`{{ url }}` Translation URL
`{{ filename }}` Transalction filename
`{{ stats }}` Translation stats, this has further attributes, examples below.
`{{ stats.all }}` Total strings count
`{{ stats.fuzzy }}` Count of strings needing review
`{{ stats.fuzzy_percent }}` Percent of strings needing review
`{{ stats.translated }}` Translated strings count
`{{ stats.translated_percent }}` Translated strings percent
`{{ stats.allchecks }}` Number of strings with failing checks
`{{ stats.allchecks_percent }}` Percent of strings with failing checks
`{{ author }}` Author of current commit, available only in the commit scope.
`{{ addon_name }}` Name of currently executed addon, available only in the addon commit message.

The following variables are available in the repository browser or editor templates:

`{{branch}}` current branch
`{{line}}` line in file
`{{filename}}` filename, you can also strip leading parts using the `parentdir` filter, for example
`{{filename|parentdir}}`

You can combine them with filters:

```
{{ component|title }}
```

You can use conditions:

```
{% if stats.translated_percent > 80 %}Well translated!{% endif %}
```

There is additional tag available for replacing chars:

```
{% replace component "-" " " %}
```

You can combine it with filters:

```
{% replace component|capfirst "-" " " %}
```

There are also additional filter to manipulate with filenames:

```

Directory of a file: {{ filename|dirname }}
File without extension: {{ filename|striptext }}
File in parent dir: {{ filename|parentdir }}
It can be used multiple times: {{ filename|parentdir|parentdir }}
  
```

...and other Django template features.

4.21.6 Importing speed

Fetching VCS repository and importing translations to Weblate can be a lengthy process, depending on size of your translations. Here are some tips:

Optimize configuration

The default configuration is useful for testing and debugging Weblate, while for a production setup, you should do some adjustments. Many of them have quite a big impact on performance. Please check *Production setup* for more details, especially:

- Configure Celery for executing background tasks (see *Background tasks using Celery*)
- *Enable caching*
- *Use a powerful database engine*
- *Disable debug mode*

Check resource limits

If you are importing huge translations or repositories, you might be hit by resource limitations of your server.

- Check the amount of free memory, having translation files cached by the operating system will greatly improve performance.
- Disk operations might be bottleneck if there is a lot of strings to process - the disk is pushed by both Weblate and the database.
- Additional CPU cores might help improve performance of background tasks (see *Background tasks using Celery*).

Disable unneeded checks

Some quality checks can be quite expensive, and if not needed, can save you some time during import if omitted. See *CHECK_LIST* for more info on how to configure this.

4.21.7 Automatic creation of components

In case your project has dozen of translation files (e.g. for different Gettext domains, or parts of Android apps), you might want to import them automatically. This can either be achieved from the command line by using *import_project* or *import_json*, or by installing the *Component discovery* addon.

To use the addon, you first need to create a component for one translation file (choose the one that is the least likely to be renamed or removed in future), and install the addon on this component.

For the management commands, you need to create a project which will contain all components and then run *import_project* or *import_json*.

See also:

Management commands, Component discovery

4.21.8 Fulltext search

Fulltext search is based on Whoosh. It is processed in the background if Celery is set up. This leads to faster site response, and a less fragmented index with the added cost that it might be slightly outdated.

See also:

Fulltext search is too slow, I get “Lock Error” quite often while translating, Rebuilding index has failed with “No space left on device”

4.22 Language definitions

To properly present different translations, Weblate needs some info about languages used. Currently definitions for about 350 languages are included, and the definition includes language name, text direction, plural definitions and language code.

4.22.1 Parsing language codes

While parsing translations, Weblate attempts to map language code (usually the ISO 639-1 one) to any existing language object. If no exact match can be found, an attempt will be made to best fit into an existing language (e.g. ignoring default country code for a given language - choosing `cs` instead of `cs_CZ`). Should also fail, a new language definition will be created using the defaults (left to right text direction, one plural) and naming of the language `:guilabel:xx_XX (generated)`. You might want to change this in the admin interface (see [Changing language definitions](#)) and report it to the issue tracker (see [Contributing](#)).

4.22.2 Changing language definitions

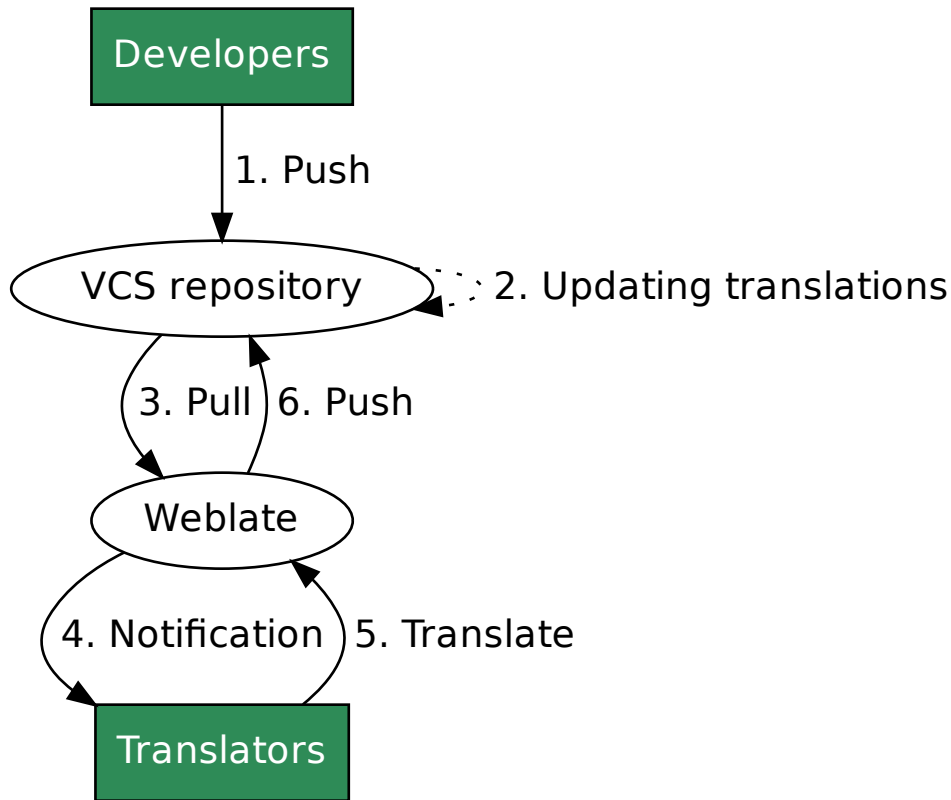
You can change language definitions in the admin interface (see [Django admin interface](#)). The *Weblate languages* section allows changing or adding language definitions. While editing, make sure all fields are correct (especially plurals and text direction), otherwise translators be unable to properly edit those translations.

4.23 Continuous localization

There is infrastructure in place so that your translation closely follows development. This way translators can work on translations the entire time, instead of working through huge amount of new text just prior to release.

This is the process:

1. Developers make changes and push them to the VCS repository.
2. Optionally the translation files are updated (this depends on the file format, see [Why does Weblate still show old translation strings when I've updated the template?](#)).
3. Weblate pulls changes from the VCS repository, see [Updating repositories](#).
4. Once Weblate detects changes in translations, translators are notified based on their subscription settings.
5. Translators submit translations using the Weblate web interface, or upload offline changes.
6. Once the translators are finished, Weblate commits the changes to the local repository (see [Lazy commits](#)) and pushes them back if it has permissions to do so (see [Pushing changes](#)).



4.23.1 Updating repositories

You should set up some way of updating backend repositories from their source. Either use hooks (see [Notification hooks](#)) or just regularly run `updategit` (with selection of project or `-all` to update all).

Whenever Weblate updates the repository, the *Post-update script* hooks are executed.

Avoiding merge conflicts

To avoid merge conflicts, control when translation files are updated in the upstream repository to avoid Weblate having changes on the same file.

You can achieve this using [Weblate's Web API](#) to force Weblate to push all pending changes and lock the translation while you are doing changes on your side.

The script for doing updates can look like this:

```
# Lock Weblate translation
wlc lock
# Push changes from Weblate to upstream repository
wlc push
```

(continues on next page)

(continued from previous page)

```
# Pull changes from upstream repository to your local copy
git pull
# Update translation files, this example is for Django
./manage.py makemessages --keep-pot -a
git commit -m 'Locale updates' -- locale
# Push changes to upstream repository
git push
# Tell Weblate to pull changes (not needed if Weblate follows your repo
# automatically)
wlc pull
# Unlock translations
wlc unlock
```

If you have multiple components sharing same repository, you need to lock them all separately:

```
wlc lock foo/bar
wlc lock foo/baz
wlc lock foo/baj
```

Note: The example uses *Weblate Client*, which needs configuration (API keys) to be able to control Weblate remotely. You can also achieve this using any HTTP client instead of wlc, e.g. curl, see *Weblate's Web API*.

Automatically receiving changes from GitHub

Weblate comes with native support for GitHub.

If you are using Hosted Weblate, the recommended approach is to install the *Hosted Weblate app*, that way you will get the correct setup without having to set much up. It can also be used for pushing changes back.

To receive notifications on every push to a GitHub repository, add the Weblate Webhook in the repository settings (*Webhooks*) as shown on the image below:

For the payload URL, append `/hooks/github/` to your Weblate URL, for example for the Hosted Weblate service, this is `https://hosted.weblate.org/hooks/github/`.

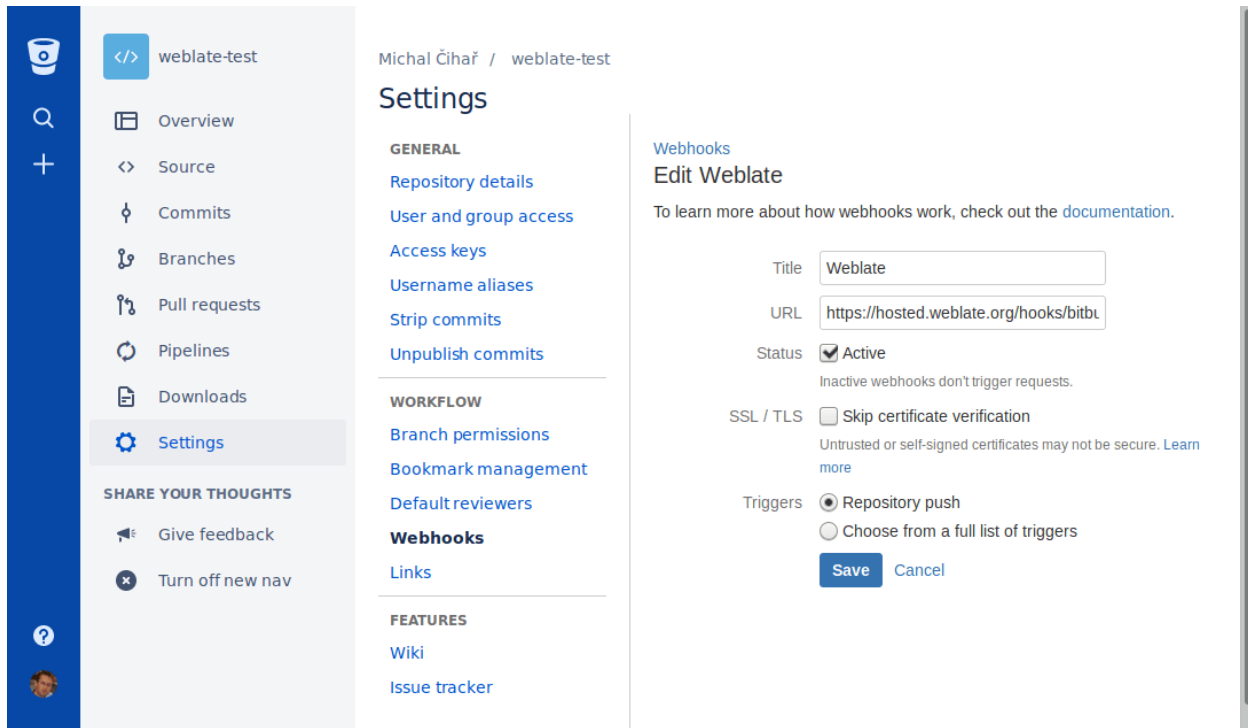
You can leave other values at default settings (Weblate can handle both content types and consumes just the *push* event).

See also:

POST /hooks/github/, Pushing changes from Hosted Weblate

Automatically receiving changes from Bitbucket

Weblate has support for Bitbucket webhooks, add a webhook which triggers upon repository push, with destination to `/hooks/bitbucket/` URL on your Weblate installation (for example `https://hosted.weblate.org/hooks/bitbucket/`).

**See also:**

POST /hooks/bitbucket/, Pushing changes from Hosted Weblate

Automatically receiving changes from GitLab

Weblate has support for GitLab hooks, add a project webhook with destination to `/hooks/gitlab/` URL on your Weblate installation (for example `https://hosted.weblate.org/hooks/gitlab/`).

See also:

POST /hooks/gitlab/, Pushing changes from Hosted Weblate

Automatically receiving changes from Pagure

New in version 3.3.

Weblate has support for Pagure hooks, add a webhook with destination to `/hooks/pagure/` URL on your Weblate installation (for example `https://hosted.weblate.org/hooks/pagure/`). This can be done in *Activate Web-hooks* under *Project options*:

The screenshot shows the Weblate interface for a project named 'nijel-test'. The top navigation bar includes the Fedora logo, the word 'PAGURE', and buttons for 'Browse', 'Create', and a user profile icon. Below this, the project name 'nijel-test' is displayed, followed by buttons for 'New Issue', 'Open PR', 'Fork', and 'Clone'. A secondary navigation bar contains links for 'Source', 'Issues' (0), 'Pull Requests' (0), 'Stats', and 'Settings' (which is the active tab).

The 'Settings' tab is divided into two main sections: 'Project Settings' on the left and 'Project Options' on the right.

Project Settings (left sidebar):

- Project Details
- Default Branch
- Private Web Hook Key
- API Keys
- Project Options** (selected)
- Public Notifications
- Users & Groups
- Deploy Keys
- Hooks
- Priorities
- Roadmap
- Close Status
- Custom Issue Fields
- Reports
- Tags
- Quick Replies
- Regenerate Repos
- Give Project
- Delete Project

Project Options (main content area):

- ☐ Activate always merge
- ☐ Activate disable non fast-forward merges
- ☐ Activate Enforce signed-off commits in pull-request
- ☒ Activate fedmsg notifications
- ☒ Activate issue tracker
- ☐ Activate issue tracker read only
- ☐ Activate issues default to private
- Activate Minimum score to merge pull-request:
- ☐ Activate notify on commit flag
- ☐ Activate notify on pull-request flag
- ☐ Activate Only assignee can merge pull-request
- ☐ Activate open metadata access to all
- ☐ Activate project documentation
- ☐ Activate pull request access only
- ☒ Activate pull requests
- ☒ Activate stomp notifications
- Activate Web-hooks:
-
-
- Learn more about
 - Flags
 - Tracker read-only
 - Pull-request access only
 - Roadmap on Issue page
 - fedmsg notifications

See also:

POST /hooks/pagure/, Pushing changes from Hosted Weblate

Automatically updating repositories nightly

Weblate automatically fetches remote repositories nightly to improve performance when merging changes later. You can optionally turn this into doing nightly merges as well, by enabling `AUTO_UPDATE`.

4.23.2 Pushing changes

Each translation component can have a push URL set up (see *Component configuration*), and in that case Weblate will be able to push change to the remote repository. Weblate can be also be configured to automatically push changes on every commit (this is default, see *Component configuration*). If you do not want changes to be pushed automatically, you can do that manually under *Repository maintenance* or using API via `wlc push`.

If you are using SSH to push, you will need to have a key without a passphrase (or use ssh-agent for Django), and the remote server needs to be verified by you via the admin interface first, otherwise pushing will fail.

The push options differ based on the *Version control integration* used, more details are found in that chapter.

Note: You can also enable automatic pushing of changes on commits, this can be done in *Component configuration*.

See also:

See *Accessing repositories* for setting up SSH keys, and *Lazy commits* for info about when Weblate decides to commit changes.

Pushing changes from Hosted Weblate

For Hosted Weblate there is a dedicated push user registered on GitHub, Bitbucket and GitLab (with username *weblate* named *Weblate push user*). You need to add this user as a collaborator and give it permission to push to your repository.

The user is added to the repository (in some cases this happens immediately, on GitHub it typically happens after accepting invitations what happens automatically every hour), you can configure your component push URL to a ssh URL of your repository (see *Component configuration*) and enjoy Weblate automatically pushing changes to your repository.

In case you do not want direct pushes by Weblate, there is support for GitHub pull requests or Gerrit reviews, you can activate these by choosing *GitHub* or *Gerrit* as VCS in *Component configuration*.

Protected branches

If you are using Weblate on protected branch, you can configure it to use pull requests and perform actual review on the translations (what might be problematic for languages you do not know). Alternative approach is to waive this limitation for the Weblate push user.

For example on GitHub this can be done in the repository configuration:

☒ **Require pull request reviews before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

Required approving reviews: 1 ▼

☐ **Dismiss stale pull request approvals when new commits are pushed**

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☐ **Require review from Code Owners**

Require an approved review in pull requests including files with a designated code owner.

☒ **Restrict who can dismiss pull request reviews**

Specify people or teams allowed to dismiss pull request reviews.

🔍 Search for people or teams

People and teams that can dismiss reviews.**Organization and repository administrators**

These members can always dismiss.

**weblate**

Weblate push user



4.23.3 Merge or rebase

By default, Weblate merges the upstream repository into its own. This is the safest way in case you also access the underlying repository by other means. In case you don't need this, you can enable rebasing of changes on upstream, which will produce history with fewer merge commits.

Note: Rebasing can cause you trouble in case of complicated merges, so carefully consider whether or not you want to enable them.

4.23.4 Interacting with others

Weblate makes it easy to interact with others using its API.

See also:

[Weblate's Web API](#)

4.23.5 Lazy commits

The behaviour of Weblate is to group commits from the same author into one commit if possible. This greatly reduces the number of commits, however you might need to explicitly tell it to do the commits in case you want to get the VCS repository in sync, e.g. for merge (this is by default allowed for the Managers group, see [Access control](#)).

The changes in this mode are committed once any of the following conditions are fulfilled:

- Somebody else changes an already changed string.
- A merge from upstream occurs.
- An explicit commit is requested.
- Change is older than period defined as *Age of changes to commit* on *Component configuration*.

If you want to commit changes more frequently and without checking of age, you can schedule a regular task to perform a commit:

```
CELERY_BEAT_SCHEDULE = {
    # Unconditionally commit all changes every 2 minutes
    "commit": {
        "task": "weblate.trans.tasks.commit_pending",
        # Ommiting hours will honor per component settings,
        # otherwise components with no changes older than this
        # won't be committed
        "kwargs": {"hours": 0},
        # How frequently to execute the job in seconds
        "schedule": 120,
    }
}
```

4.23.6 Processing repository with scripts

The way to customize how Weblate interacts with the repository is *Addons*. Consult *Executing scripts from addon* for info on how to execute external scripts through addons.

4.24 Licensing translations

Weblate allows you to specify under which license the translations are contributed. This is especially important to specify if the translations are open to the public to raise proper expectations what can be done with the translations.

There are two things you specify on the *Component configuration* - license information and the contributor agreement.

4.24.1 License information

Upon specifying license information (license name and URL), this information is shown in the translation information, but it is not enforced in any way.

Usually this is best location to place information on licensing where no explicit consent is required.

4.24.2 Contributor agreement

Once you specify contributor agreement, only users who have agreed to it will be able to contribute. This is clearly visible when accessing the translation:

The entered text is formatted into paragraphs and external links are possible. HTML markup can not be used.

4.24.3 Signed off by

Should your project require Signed-off-by header in the commits, you should enable contributor agreement with the DCO text and add the header to the commit message (see [Template markup](#) for more details). The full commit message can look like:

```
Translated using Weblate ({{ language_name }})

Currently translated at {{ stats.translated_percent }}% ({{ stats.translated }} of {{ stats.all }} strings)

Translation: {{ project_name }}/{{ component_name }}
Translate-URL: {{ url }}
Signed-off-by: {{ author }}
```

4.24.4 User licenses

User can review licenses on projects he is contributing to in the profile:

Licenses

Please pay attention to the licensing information as these specify how others are allowed to use your translations.

By registering you agree to use your name and email in the commits and provide your contribution under license defined by each translated project.

You have agreed to the following as a contributor:

- [WeblateOrg/Language names](#)

Following translations have explicitly specified their licensing and copyright conditions:

Project	License
WeblateOrg/Android	MIT
WeblateOrg/Django	GPL-3.0+
WeblateOrg/Djangojs	GPL-3.0+
WeblateOrg/Language names	GPL-3.0+

Powered by Weblate 3.7.1 About Weblate Legal Contact Documentation Donate to Weblate

4.25 Translation process

4.25.1 Suggestion voting

New in version 1.6: This feature is available since Weblate 1.6.

By default, everyone can add suggestions, which logged in users can accept. Requiring more than one person for acceptance can be achieved by suggestion voting. You can enable this on *Component configuration* configuration by *Suggestion voting* and *Autoaccept suggestions*. The first one enables the voting feature, while the latter sets the threshold a suggestion is automatically accepted (this includes a vote from the user making the suggestion).

Note: Once automatic acceptance is set up, normal users lose the privilege to directly save translations or accept suggestions. This can be overridden by *Can override suggestion state* privilege (see *Access control*).

You can combine these with *Access control* into one of the following setups:

- Users suggest and vote for suggestions, a limited group controls what is accepted - turn on voting, but automatic acceptance off, and don't let users save translations.
- Users suggest and vote for suggestions with automatic acceptance once the defined number of them agree - turn on voting and set the desired number of votes for automatic acceptance.
- Optional voting for suggestions - you can also turn on voting only, and in this case it can optionally be used by users when they are unsure about a translation by making multiple suggestions.

4.25.2 Additional info on source strings

Enhance the translation process with info available in the translation files. This includes string prioritization, check flags, or providing visual context. All these features can be set on the *Reviewing source strings*:

Weblate
Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Django / source strings / review

Source

Files

Priority

Medium

Failing checks

Language	Status	Checks	Edit
Czech	✓		Edit
Hebrew	✓		Edit
Hungarian	✓		Edit

Translation flags

Add

Please enter a comma separated list of translation flags, see [documentation](#) for more details.

Save

Priority

Medium

Higher priority strings are presented to translators earlier.

Save

Additional context

Save

Screenshot context

No screenshot currently associated! [Manage screenshots](#)

Add new screenshot

Screenshot name

Image

Browse...

No file selected.

Upload JPEG or PNG images up to 2000x2000 pixels.

Upload

Source string location

[weblate/templates/translation.html:45](#)
[weblate/trans/forms.py:1404](#)

Source string age

3 minutes ago

Translation file

weblate/locale/cs/LC_MESSAGES/django.po, string 1

Powered by Weblate 3.7.1
About Weblate
Legal
Contact
Documentation
Donate to Weblate

Access this directly from the translating interface by clicking the “Edit” icon next to *Screenshot context*, *Flags* or *String priority*:

New in version 2.0.

Translation flags

4.25. Translation process

Changed in version 3.3: Previously this was called *Quality checks flags*, but as it no longer configures only checks, the name was changed to be more generic.

The default set of translation flags is determined by the translation *Component configuration* and the translation file. However, you might want to use it to customize this per source string.

See also:

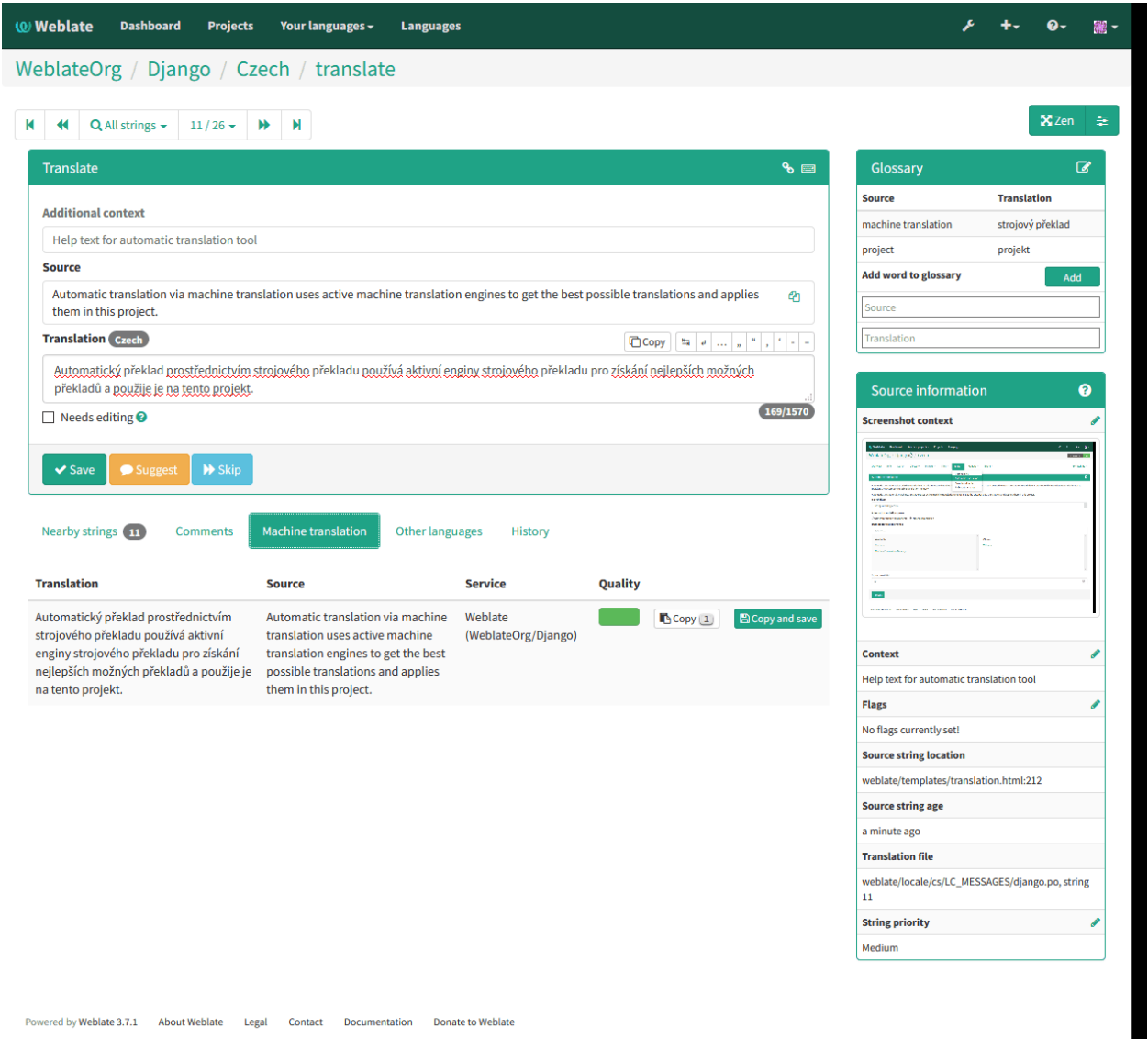
Quality checks

Visual context for strings

New in version 2.9.

You can upload a screenshot showing a given source string in use within your program. This helps translators understand where it is used, and how it should be translated.

The uploaded screenshot is shown in the translation context sidebar:



In addition to *Reviewing source strings*, screenshots have a separate management interface under *Tools* menu. Upload screenshots, assign them to source strings manually or with the use of OCR.

Once a screenshot is uploaded, this interface handles management and assigning it to source strings:

Source string	Actions
No source strings are currently assigned!	
Screenshot is shown to add visual context for all listed source strings.	

Source string	Context	Location	Actions
Files		weblate/templates/translation.html:45, weblate/trans/forms.py:1404	+ Add to screenshot
Automatic translation via machine translation uses active machine translation engines to get the best possible translations and applies them in this project.		weblate/templates/translation.html:212	+ Add to screenshot

[🔍 Search](#)[⚙️ Automatically recognize](#)

Webate

Dashboard

Watched projects

Projects

Languages

🔍

+

🔔

👤

WebateOrg / Django / Czech

translated 96%

Overview

Info

Search

Glossary

Insights

Files

Tools

Manage

Share

Watching

Automatic translation

?

Automatic translation takes existing translations in this project and applies them to the new component. It can be used to push translations to a different branch, to fix inconsistent translations or to translate a new component using translation memory.

Automatic translation via machine translation uses active machine translation engines to get the best possible translations and applies them in this project.

Search filter

Strings needing action

Automatic translation source

☐ Other translation components ☒ Machine translation

Machine translation engines

Search...

Available:

Webate

Webate Translation Memory

Chosen:

Webate

Score threshold

80

Process

Powered by Webate 3.7

About Webate

Legal

Contact

Documentation

Donate to Webate

192

Screenshot name

Automatic translation

Image

Currently: [screenshots/automatic-translation.png](#)
Change:

Chapter 4. Licen

4.26 Checks and fixups

4.26.1 Custom automatic fixups

You can also implement your own automatic fixup in addition to the standard ones and include them in `AUTOFIX_LIST`.

The automatic fixes are powerful, but can also cause damage; be careful when writing one.

For example, the following automatic fixup would replace every occurrence of string `foo` in translation with `bar`:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#

from django.utils.translation import ugettext_lazy as _

from weblate.trans.autofixes.base import AutoFix

class ReplaceFooWithBar(AutoFix):
    """Replace foo with bar."""

    name = _("Foobar")

    def fix_single_target(self, target, source, unit):
        if "foo" in target:
            return target.replace("foo", "bar"), True
        return target, False
```

To install custom checks, you need to provide a fully-qualified path to the Python class in the `AUTOFIX_LIST`, see *Custom quality checks and auto fixes*.

4.26.2 Customizing behavior

You can fine tune Weblate behavior (mostly checks) for each source string (in source strings review, see *Additional info on source strings*) or in the *Component configuration (Translation flags)*. Some file formats also allow to specify flags directly in the format.

Here is a list of flags currently accepted:

rst-text Treat text as RST document, effects *Unchanged translation*.

md-text Treat text as Markdown document.

dos-eol Use DOS end of line markers instead of Unix ones (`\r\n` instead of `\n`).

url The string should consist of URL only.

max-length:N Limit maximal length for string to N chars, see *Maximum Length*

xml-text Treat text as XML document, affects *Invalid XML markup* and *XML tags mismatch*.

font-family:NAME Define font family for rendering checks, see *Managing fonts*.

font-weight:WEIGHT Define font weight for rendering checks, see *Managing fonts*.

font-size:SIZE Define font size for rendering checks, see *Managing fonts*.

font-spacing:SPACING Define font spacing for rendering checks, see *Managing fonts*.

python-format, c-format, php-format, python-brace-format, javascript-format, c-sharp-format, java-format Treats all strings like format strings, affects *Formatted strings*, *Formatted strings*, *Formatted strings*, *Formatted strings*, *Formatted strings*, *Formatted strings*, *Formatted strings*, *Unchanged translation*.

ignore-end-space Skip the “Trailing space” quality check.

ignore-inconsistent Skip the “Inconsistent” quality check.

ignore-translated Skip the “Has been translated” quality check.

ignore-begin-newline Skip the “Starting newline” quality check.

ignore-zero-width-space Skip the “Zero-width space” quality check.

ignore-escaped-newline Skip the “Mismatched n” quality check.

ignore-same Skip the “Unchanged translation” quality check.

ignore-end-question Skip the “Trailing question” quality check.

ignore-end-ellipsis Skip the “Trailing ellipsis” quality check.

ignore-ellipsis Skip the “Ellipsis” quality check.

ignore-python-brace-format Skip the “Python brace format” quality check.

ignore-end-newline Skip the “Trailing newline” quality check.

ignore-c-format Skip the “C format” quality check.

ignore-javascript-format Skip the “JavaScript format” quality check.

ignore-optional-plural Skip the “Optional plural” quality check.

ignore-end-exclamation Skip the “Trailing exclamation” quality check.

ignore-end-colon Skip the “Trailing colon” quality check.

ignore-xml-invalid Skip the “Invalid XML markup” quality check.

ignore-xml-tags Skip the “XML tags mismatch” quality check.

ignore-python-format Skip the “Python format” quality check.

ignore-plurals Skip the “Missing plurals” quality check.

ignore-begin-space Skip the “Starting spaces” quality check.

ignore-bbcode Skip the “Mismatched BBcode” quality check.

ignore-multiple-failures Skip the “Multiple failing checks” quality check.

ignore-php-format Skip the “PHP format” quality check.

ignore-end-stop Skip the “Trailing stop” quality check.

ignore-angularjs-format Skip the “AngularJS interpolation string” quality check.

ignore-c-sharp-format Skip the “C# format” quality check.

ignore-java-format Skip the “Java format” quality check.

Note: Generally the rule is named `ignore-*` for any check, using its identifier, so you can use this even for your custom checks.

These flags are understood both in *Component configuration* settings, per source string settings and in translation file itself (eg. in GNU Gettext).

4.26.3 Managing fonts

New in version 3.7.

The *Maximum size of translation* check needs fonts to properly calculate dimensions of rendered text. The fonts can be managed in Weblate in the font management tool which you can find as *Fonts* under *Tools* menu of your translation project.

You can upload TrueType or OpenType fonts, configure font groups and use those in the the check.

The font groups allow you to define different fonts for different languages, what is typically needed for non latin languages:

[Weblate](#) [Dashboard](#) [Projects](#) [Your languages](#) [Languages](#)

WeblateOrg / Font groups / default-font

Font group

Name	default-font		
Default font	Source Sans Pro Bold		
Japanese	language override	Droid Sans Fallback Regular	Remove
Korean	language override	Droid Sans Fallback Regular	Remove
Delete			

Add language override

Language

Font

[Save](#)

Edit font group

Font group name

default-font

Identifier you will use in checks to select this font group. Avoid whitespace or special chars.

Default font

Source Sans Pro Bold

Default font is used unless per language override matches.

[Save](#)

Powered by Weblate 3.7.1 [About Weblate](#) [Legal](#) [Contact](#) [Documentation](#) [Donate to Weblate](#)

The font groups are identified by name, which can not contain whitespace or special chars to be easy to use in check definition:

[Dashboard](#)
[Projects](#)
[Your languages ▾](#)
[Languages](#)

[WeblateOrg](#) / [Fonts](#)

[Font groups](#)
[Fonts](#)

Group name ▾	Default font ▾	Language overrides ▾	
default-font	Source Sans Pro Bold	Japanese: Droid Sans Fallback Regular Korean: Droid Sans Fallback Regular	Edit

Add font group

Font group name

Identifier you will use in checks to select this font group. Avoid whitespace or special chars.

Default font

Default font is used unless per language override matches.

[Save](#)

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

After upload the font family and style is automatically recognized:

[Dashboard](#)
[Projects](#)
[Your languages ▾](#)
[Languages](#)

[WeblateOrg](#) / [Fonts](#) / [Droid Sans Fallback Regular](#)

Font

Font family	Droid Sans Fallback
Font style	Regular
File size	3939852
Created on	now
Uploaded by	testuser
Used in groups	

[Delete](#)

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

You can have number of fonts loaded into Weblate:

4.26.4 Writing own checks

Weblate comes with wide range of quality checks (see [Quality checks](#)), though they might not 100% cover all you want to check. The list of performed checks can be adjusted using `CHECK_LIST` and you can also add custom checks. All you need to do is to subclass `weblate.checks.Check`, set few attributes and implement either `check` or `check_single` methods (first one if you want to deal with plurals in your code, the latter one does this for you). You will find below some examples.

To install custom checks, you need to provide a fully-qualified path to the Python class in the `CHECK_LIST`, see [Custom quality checks and auto fixes](#).

Checking translation text does not contain “foo”

This is a pretty simple check which just checks whether translation does not contain string “foo”.

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#
```

(continues on next page)

(continued from previous page)

```

"""Simple quality check example."""

from django.utils.translation import ugettext_lazy as _

from weblate.checks.base import TargetCheck

class FooCheck(TargetCheck):

    # Used as identifier for check, should be unique
    # Has to be shorter than 50 chars
    check_id = "foo"

    # Short name used to display failing check
    name = _("Foo check")

    # Description for failing check
    description = _("Your translation is foo")

    # Real check code
    def check_single(self, source, target, unit):
        return "foo" in target

```

Checking Czech translation text plurals differ

Check using language information to verify that two plural forms in Czech language are not same.

```

# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#
"""Quality check example for Czech plurals."""

from django.utils.translation import ugettext_lazy as _

from weblate.checks.base import TargetCheck

class PluralCzechCheck(TargetCheck):

```

(continues on next page)

(continued from previous page)

```
# Used as identifier for check, should be unique
# Has to be shorter than 50 chars
check_id = "foo"

# Short name used to display failing check
name = _("Foo check")

# Description for failing check
description = _("Your translation is foo")

# Real check code
def check_target_unit(self, sources, targets, unit):
    if self.is_language(unit, ("cs",)):
        return targets[1] == targets[2]
    return False

def check_single(self, source, target, unit):
    """We don't check target strings here."""
    return False
```

4.27 Machine translation

Weblate has built in support for several machine translation services and it's up to the administrator to enable them. The services have different terms of use, so please check whether you are allowed to use them before enabling them in Weblate. The individual services are enabled using *MT_SERVICES*.

The source language can be configured at *Project configuration*.

4.27.1 Amagama

Special installation of *tmserver* run by Virtaal authors.

To enable this service, add `weblate.machinery.tmserver.AmagamaTranslation` to *MT_SERVICES*.

See also:

[Amagama](#), [Amagama Translation Memory](#)

4.27.2 Apertium

A free/open-source machine translation platform providing translation to a limited set of languages.

The recommended way to use Apertium is to run your own Apertium APy server.

To enable this service, add `weblate.machinery.apertium.ApertiumAPYTranslation` to *MT_SERVICES* and set *MT_APERTIUM_APY*.

See also:

MT_APERTIUM_APY, [Apertium website](#), [Apertium APy documentation](#)

4.27.3 AWS

New in version 3.1.

Amazon Translate is a neural machine translation service for translating text to and from English across a breadth of supported languages.

To enable this service, add `weblate.machinery.aws.AWSTranslation` to `MT_SERVICES`, install the `boto3` module and set the settings.

See also:

`MT_AWS_REGION`, `MT_AWS_ACCESS_KEY_ID`, `MT_AWS_SECRET_ACCESS_KEY` [Amazon Translate Documentation](#)

4.27.4 Baidu API machine translation

New in version 3.2.

Machine translation service provided by Baidu.

This service uses an API and you need to obtain ID and API key from Baidu.

To enable this service, add `weblate.machinery.baidu.BaiduTranslation` to `MT_SERVICES` and set `MT_BAIDU_ID` and `MT_BAIDU_SECRET`.

See also:

`MT_BAIDU_ID`, `MT_BAIDU_SECRET` [Baidu Translate API](#)

4.27.5 DeepL

New in version 2.20.

DeepL is paid service providing good machine translation for few languages. According to some benchmark it's currently best available service.

To enable this service, add `weblate.machinery.deepl.DeepLTranslation` to `MT_SERVICES` and set `MT_DEEPL_KEY`.

See also:

`MT_DEEPL_KEY`, [DeepL website](#), [DeepL API documentation](#)

4.27.6 Glosbe

Free dictionary and translation memory for almost every living language.

API is free to use, but subject to the used data source license. There is a limit of calls that may be done from one IP in fixed period of time, to prevent abuse.

To enable this service, add `weblate.machinery.glosbe.GlosbeTranslation` to `MT_SERVICES`.

See also:

[Glosbe website](#)

4.27.7 Google Translate

Machine translation service provided by Google.

This service uses Translation API and you need to obtain an API key and enable billing on Google API console.

To enable this service, add `weblate.machinery.google.GoogleTranslation` to `MT_SERVICES` and set `MT_GOOGLE_KEY`.

See also:

`MT_GOOGLE_KEY`, [Google translate documentation](#)

4.27.8 Microsoft Cognitive Services Translator

New in version 2.10.

Machine translation service provided by Microsoft in Azure portal as a one of Cognitive Services.

You need to register at Azure portal and use the key you obtain there.

To enable this service, add `weblate.machinery.microsoft.MicrosoftCognitiveTranslation` to `MT_SERVICES` and set `MT_MICROSOFT_COGNITIVE_KEY`.

See also:

`MT_MICROSOFT_COGNITIVE_KEY`, [Cognitive Services - Text Translation API](#), [Microsoft Azure Portal](#)

4.27.9 Microsoft Terminology Service

New in version 2.19.

The Microsoft Terminology Service API allows you to programmatically access the terminology, definitions and user interface (UI) strings available on the Language Portal through a web service.

To enable this service, add `weblate.machinery.microsoftterminology.MicrosoftTerminologyService` to `MT_SERVICES`.

See also:

[Microsoft Terminology Service API](#)

4.27.10 MyMemory

Huge translation memory with machine translation.

Free, anonymous usage is currently limited to 100 requests/day, or to 1000 requests/day when you provide contact email in `MT_MYMEMORY_EMAIL`. You can also ask them for more.

To enable this service, add `weblate.machinery.mymemory.MyMemoryTranslation` to `MT_SERVICES` and set `MT_MYMEMORY_EMAIL`.

See also:

`MT_MYMEMORY_EMAIL`, `MT_MYMEMORY_USER`, `MT_MYMEMORY_KEY`, [MyMemory website](#)

4.27.11 Netease Sight API machine translation

New in version 3.3.

Machine translation service provided by Netease.

This service uses an API and you need to obtain key and secret from Netease.

To enable this service, add `weblate.machinery.youdao.NeteaseSightTranslation` to `MT_SERVICES` and set `MT_NETEASE_KEY` and `MT_NETEASE_SECRET`.

See also:

`MT_NETEASE_KEY`, `MT_NETEASE_SECRET` Netease Sight Translation Platform

4.27.12 tmserver

You can run your own translation memory server which is bundled with Translate-toolkit and let Weblate talk to it. You can also use it with amaGama server, which is an enhanced version of tmserver.

First you will want to import some data to the translation memory:

To enable this service, add `weblate.machinery.tmserver.TMServerTranslation` to `MT_SERVICES`.

```
build_tmdb -d /var/lib/tm/db -s en -t cs locale/cs/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t de locale/de/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t fr locale/fr/LC_MESSAGES/django.po
```

Now you can start tmserver to listen to your requests:

```
tmserver -d /var/lib/tm/db
```

And configure Weblate to talk to it:

```
MT_TMSERVER = 'http://localhost:8888/tmserver/'
```

See also:

`MT_TMSERVER`, tmserver Amagama, Amagama Translation Memory

4.27.13 Yandex Translate

Machine translation service provided by Yandex.

This service uses Translation API and you need to obtain API key from Yandex.

To enable this service, add `weblate.machinery.yandex.YandexTranslation` to `MT_SERVICES` and set `MT_YANDEX_KEY`.

See also:

`MT_YANDEX_KEY`, Yandex Translate API, Powered by Yandex.Translate

4.27.14 Youdao Zhiyun API machine translation

New in version 3.2.

Machine translation service provided by Youdao.

This service uses an API and you need to obtain ID and API key from Youdao.

To enable this service, add `weblate.machinery.youdao.YoudaoTranslation` to `MT_SERVICES` and set `MT_YOUDAO_ID` and `MT_YOUDAO_SECRET`.

See also:

`MT_YOUDAO_ID`, `MT_YOUDAO_SECRET` Youdao Zhiyun Natural Language Translation Service

4.27.15 Weblate

Weblate can be source of machine translation as well. It is based on the fulltext engine Whoosh and provides both exact and inexact matches.

To enable these services, add `weblate.machinery.weblatetm.WeblateTranslation` to `MT_SERVICES`.

4.27.16 Weblate Translation Memory

New in version 2.20.

The *Translation Memory* can be used as source for machine translation suggestions as well.

To enable these services, add `weblate.memory.machine.WeblateMemory` to the `MT_SERVICES`. This service is enabled by default.

4.27.17 SAP Translation Hub

Machine translation service provided by SAP.

You need to have a SAP account (and enabled the SAP Translation Hub in the SAP Cloud Platform) to use this service.

To enable this service, add `weblate.machinery.saptranslationhub.SAPTranslationHub` to `MT_SERVICES` and set appropriate access to either sandbox or productive API.

Note: To access the Sandbox API, you need to set `MT_SAP_BASE_URL` and `MT_SAP_SANDBOX_APIKEY`.

To access the productive API, you need to set `MT_SAP_BASE_URL`, `MT_SAP_USERNAME` and `MT_SAP_PASSWORD`.

See also:

`MT_SAP_BASE_URL`, `MT_SAP_SANDBOX_APIKEY`, `MT_SAP_USERNAME`, `MT_SAP_PASSWORD`, `MT_SAP_USE_MT` SAP Translation Hub API

4.27.18 Custom machine translation

You can also implement your own machine translation services using a few lines of Python code. This example implements translation to a fixed list of languages using dictionary Python module:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
```

(continues on next page)

(continued from previous page)

```
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#
"""Machine translation example."""

import dictionary
from weblate.machinery.base import MachineTranslation

class SampleTranslation(MachineTranslation):
    """Sample machine translation interface."""

    name = "Sample"

    def download_languages(self):
        """Return list of languages your machine translation supports."""
        return {"cs"}

    def download_translations(self, source, language, text, unit, user):
        """Return tuple with translations."""
        return [(t, 100, self.name, text) for t in dictionary.translate(text)]
```

You can list own class in `MT_SERVICES` and Weblate will start using that.

4.28 Addons

New in version 2.19.

Addons provide ways to customize translation workflow. You can install addons to your translation component and they will work behind the scenes. The addon management can be found under *Manage* menu of a translation component.

[Dashboard](#)
[Watched projects](#)
[Projects](#)
[Your languages](#)
[Languages](#)

[WeblateOrg](#) / [Language names](#) / [Addons](#)

Installed addons

There are no addons currently installed.

Available addons

Language consistency

project wide

This addon ensures that all components within one project have translation to same languages.

+ Install

Component discovery

repository wide

This addon automatically adds or removes components to the project based on file changes in the version control system.

+ Install

Statistics generator

This addon generates a file containing detailed information about the translation.

+ Install

Contributors in comment

Update comment in the PO file header to include contributor name and years of contributions.

+ Install

Customize gettext output

Allows customization of gettext output behavior, for example line wrapping.

+ Install

Generate MO files

Automatically generates MO file for every changed PO file.

+ Install

Update PO files to match POT (msgmerge)

Update all PO files to match the POT file using msgmerge. This is triggered whenever new changes are pulled from the upstream repository.

+ Install

Squash Git commits

repository wide

Squash Git commits prior to pushing changes.

+ Install

Stale comment removal

project wide

Set timeframe for removal of comments.

+ Install

Stale suggestion removal

project wide

Set timeframe for removal of suggestions.

+ Install

Some addons will ask for additional configuration during installation.

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

4.28.1 Built in addons

Cleanup translation files

Update all translation files to match the monolingual base file. For most file formats, this means removing stale translation keys no longer present in the base file.

Language consistency

This addon ensures that all components within one project have translation to same languages.

Unlike most others, this addon operates on whole project.

Component discovery

This addon automatically adds or removes components to the project based on file changes in the version control system.

It is similar to the `import_project` management command, but the major difference is that it is triggered on every VCS update. This way you can easily track multiple translation components within one VCS.

To use component discovery, you first need to create one component which will act as master and others will use *Weblate internal URLs* to it as a VCS configuration. You should choose the one which is less likely to disappear in the future here.

Once you have one component from the target VCS, you can configure the discovery addon to find all translation components in the VCS. The matching is done using regular expression so it can be quite powerful, but it can be complex to configure. You can use examples in the addon help for some common use cases.

Once you hit save, you will be presented with a preview of matched components, so you can check whether the configuration actually matches your needs:

Weblate
Dashboard
Watched projects
Projects
Your languages
Languages

WeblateOrg / Language names / Addons / Component discovery

Configure addon

Please review and confirm matched components.

Component	Matched files
Following components would be created	
Djangojs	weblate/locale/hu/LC_MESSAGES/djangojs.po (hu) weblate/locale/cs/LC_MESSAGES/djangojs.po (cs) weblate/locale/he/LC_MESSAGES/djangojs.po (he)
Django	weblate/locale/cs/LC_MESSAGES/django.po (cs) weblate/locale/he/LC_MESSAGES/django.po (he) weblate/locale/hu/LC_MESSAGES/django.po (hu)

☐ I confirm the above matches look correct

Regular expression to match translation files

weblate/locale/(?P<language>[^\s]*/LC_MESSAGES/(?P<component>[^\s]*)\.po

File format

Gettext PO file

Customize the component name

{{ component|title }}

Define the monolingual base filename

Leave empty for bilingual translation files.

Define the base file for new translations

weblate/locale/{{ component }}.pot

Filename of file used for creating new translations. For gettext choose .pot file.

Language filter

^(cs|he|hu)\$

Regular expression to filter translation against when scanning for filemask.

☐ Remove components for inexistant files

The regular expression to match translation files has to contain two named groups to match component and language, some examples:

Regular expression	Example matched files	Description
(?P<language>[^\s]*)/(?P<component>[^\s]*)\.po	cs/application.po cs/website.po de/application.po de/website.po	One folder per language containing translation files for components.
locale/(?P<language>[^\s]*/LC_MESSAGES/(?P<component>[^\s]*)\.po	locale/cs/LC_MESSAGES/application.po locale/cs/LC_MESSAGES/website.po locale/de/LC_MESSAGES/application.po locale/de/LC_MESSAGES/website.po	Usual structure for storing gettext PO files.
src/locale/(?P<component>[^\s]*)\.(?P<language>[^\s]*)\.po	src/locale/application.cs.po src/locale/website.cs.po src/locale/application.de.po src/locale/website.de.po	Using both component and language name within filename.
locale/(?P<language>[^\s]*/(?P<component>[^\s]*)(?P=language)\.po	locale/cs/application/cs.po locale/cs/website/cs.po locale/de/application/de.po locale/de/website/de.po	Using language in both path and filename.
res/values-(?P<language>[^\s]*)/strings-(?P<component>[^\s]*)\.xml	res/values-cs/strings-about.xml res/values-cs/strings-help.xml res/values-de/strings-about.xml res/values-de/strings-help.xml	Android resource strings, split into several files.

You can use Django template markup in both component name and the monolingual base filename, for example:

`{{ component }}`
Component filename match

`{{ component|title }}`
Component filename with upper case first letter

Save

See also:

Template markup

Flag unchanged translations as “Needs editing”

New in version 3.1.

Whenever a new translatable string is imported from the VCS and it matches source strings, it is flagged as needing editing in Weblate. This is especially useful for file formats that include all strings even if they are not translated.

Flag new source strings as “Needs editing”

Whenever a new source string is imported from the VCS, it is flagged as needing editing in Weblate. This way you can easily filter and edit source strings written by the developers.

Flag new translations as “Needs editing”

Whenever a new translatable string is imported from the VCS, it is flagged as needing editing in Weblate. This way you can easily filter and edit translations created by the developers.

Statistics generator

This addon generates a file containing detailed information about the translation. You can use Django template in both filename and content, see *Template markup* for detailed markup description.

For example generating summary file for each translations:

Name of generated file `locale/{{ language_code }}.json`

Content

```
{
  "language": "{{ language_code }}",
  "strings": "{{ stats.all }}",
  "translated": "{{ stats.translated }}",
  "last_changed": "{{ stats.last_changed }}",
  "last_author": "{{ stats.last_author }}"
}
```

See also:

Template markup

Contributors in comment

Update comment in the PO file header to include contributor name and years of contributions.

Update ALL_LINGUAS variable in the configure file

Updates the ALL_LINGUAS variable in configure, configure.in or configure.ac files, when a new translation is added.

Customize gettext output

Allows customization of gettext output behavior, for example line wrapping.

Update LINGUAS file

Updates the LINGUAS file when a new translation is added.

Generate MO files

Automatically generates MO file for every changed PO file.

Update PO files to match POT (msgmerge)

Update all PO files to match the POT file using msgmerge. This is triggered whenever new changes are pulled from the upstream repository.

Squash Git commits

Squash Git commits prior to pushing changes.

Customize JSON output

Allows to customize JSON output behavior, for example indentation or sorting.

Formats the Java properties file

This addon sorts the Java properties file.

Stale comment removal

New in version 3.7.

Set timeframe for removal of comments. This can be useful to remove old comments which might have become outdated. Use with care as comment being old does not mean it has lost its importation.

Stale suggestion removal

New in version 3.7.

Set timeframe for removal of suggestions. This can be very useful in connection with suggestion voting (see [Peer review](#)) to remove suggestions which don't receive enough positive votes until certain deadline.

4.28.2 Customizing list of addons

List of addons is configured by `WEBLATE_ADDONS`, to add another addon simply include class absolute name in this setting.

4.28.3 Writing addon

You can write own addons as well, all you need to do is subclass `BaseAddon`, define addon metadata and implement callback which will do the processing.

You can look at example addon for more information:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#

from __future__ import unicode_literals

from django.utils.translation import ugettext_lazy as _

from weblate.addons.base import BaseAddon
from weblate.addons.events import EVENT_PRE_COMMIT

class ExampleAddon(BaseAddon):
    # Filter for compatible components, every key is
    # matched against property of component
    compat = {
        'file_format': frozenset((
            'po', 'po-mono',
        )),
    }
    # List of events addon should receive
    events = (EVENT_PRE_COMMIT,)
    # Addon unique identifier
    name = 'weblate.example.example'
    # Verbose name shown in the user interface
    verbose = _('Example addon')
    # Detailed addon description
    description = _('This addon does nothing it is just an example.')

    # Callback to implement custom behavior
    def pre_commit(self, translation, author):
        return
```

4.28.4 Executing scripts from addon

You can also use addons to execute external scripts. This used to be integrated in Weblate, but now you have to write little code to wrap your script with an addon.

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#
"""
Example pre commit script
"""

from __future__ import unicode_literals

from django.utils.translation import ugettext_lazy as _

from weblate.addons.events import EVENT_PRE_COMMIT
from weblate.addons.scripts import BaseScriptAddon


class ExamplePreAddon(BaseScriptAddon):
    # Event used to trigger the script
    events = (EVENT_PRE_COMMIT,)
    # Name of the addon, has to be unique
    name = 'weblate.example.pre'
    # Verbose name and long description
    verbose = _('Execute script before commit')
    description = _('This addon executes a script.')

    # Script to execute
    script = '/bin/true'
    # File to add in commit (for pre commit event)
    # does not have to be set
    add_file = 'po/{{ language_code }}.po'
```

For installing instructions see *Custom addons*.

The script is executed with the current directory set to the root of the VCS repository for given component.

Additionally, the following environment variables are available:

WL_VCS

Version control system used.

WL_REPO

Upstream repository URL.

WL_PATH

Absolute path to VCS repository.

WL_BRANCH

New in version 2.11.

Repository branch configured in the current component.

WL_FILEMASK

File mask for current component.

WL_TEMPLATE

File name of template for monolingual translations (can be empty).

WL_NEW_BASE

New in version 2.14.

File name of the file which is used for creating new translations (can be empty).

WL_FILE_FORMAT

File format used in current component.

WL_LANGUAGE

Language of currently processed translation (not available for component level hooks).

WL_PREVIOUS_HEAD

Previous HEAD on update (available only available when running post update hook).

See also:

[Component configuration](#)

Post update repository processing

Post update repository processing can be used to update translation files on the source change. To achieve this, please remember that Weblate only sees files which are committed to the VCS, so you need to commit changes as a part of the script.

For example with gulp you can do it using following code:

```
#!/bin/sh
gulp --gulpfile gulp-i18n-extract.js
git commit -m 'Update source strings' src/languages/en.lang.json
```

Pre commit processing of translations

In many cases you might want to automatically do some changes to the translation before it is committed to the repository. The pre commit script is exactly the place to achieve this.

It is passed a single parameter consisting of filename of current translation.

4.29 Translation Memory

New in version 2.20.

Weblate comes with a built-in translation memory.

The translation memory consists of following content:

- Manually imported translation memory (see *User interface*).
- Automatically stored translations performed in Weblate (depending on *Translation memory scopes*).

The translation memory can be used to get matches:

- In the *Machine translation* view while translating.
- Automatically translate strings using *Automatic translation*.

For installation tips, see *Weblate Translation Memory*, however this service is enabled by default.

4.29.1 Translation memory scopes

New in version 3.2: The different translation memory scopes are available since Weblate 3.2, prior to this release translation memory could be only loaded from file corresponding to the current imported translation memory scope.

The translation memory scopes are there to allow both privacy and sharing of translations, depending on the actual desired behavior.

Imported translation memory

You can import arbitrary translation memory data using `import_memory` command. The memory content will be available for all users and projects.

Per user translation memory

All user translations are automatically stored in personal translation memory. This memory is available only for this user.

Per project translation memory

All translations within a project are automatically stored in a project translation memory. This memory is available only for this project.

Shared translation memory

All translation within projects which have enabled shared translation memory are stored in shared translation memory. This shared memory is available for all projects then.

Please consider carefully when enabling this feature on shared Weblate installations as this might have severe implications:

- The translations can be used by anybody else.
- This might lead to disclosing secret information.

4.29.2 Managing translation memory

User interface

New in version 3.2.

There is basic user interface to manage per user and per project translation memories. It can be used to download, wipe or import it.

The downloads in JSON are useful for Weblate, TMX is provided for interoperability with other tools.

The screenshot shows the Weblate user interface for managing translation memory. At the top is a dark green navigation bar with links to 'Weblate', 'Dashboard', 'Projects', 'Your languages', and 'Languages'. Below this is a breadcrumb trail: 'testuser / Translation memory'. The main content area has a green header 'Translation memory status' with a help icon. It contains a table with two rows: 'Number of your entries' and 'Total number of entries', both showing a value of 0. Below the table are two green buttons: 'Download as JSON' and 'Download as TMX'. The next section has a green header 'Import translation memory'. It includes a 'File' section with a 'Browse...' button and the text 'No file selected.' and 'You can upload a TMX or JSON file.' Below this is an 'Upload' button. The final section has a red header 'Wipe translation memory'. It contains a checkbox labeled 'Confirm deleting all translation memory entries' which is currently unchecked. Below this is a red 'Wipe' button. At the bottom of the page is a footer with the text 'Powered by Weblate 3.7.1' and several links: 'About Weblate', 'Legal', 'Contact', 'Documentation', and 'Donate to Weblate'.

Management interface

There are several management commands to manipulate with the translation memory content, these operate on memory as whole not filtered by scopes (unless requested by parameters):

`dump_memory` Exporting the memory into JSON

`import_memory` Importing TMX or JSON files into the memory

`list_memory` Listing memory content

`delete_memory` Deleting content from the memory

4.30 Configuration

All settings are stored in `settings.py` (as usual for Django).

Note: After changing any of these settings, you need to restart Weblate. In case it is run as `mod_wsgi`, you need to restart Apache to reload the configuration.

See also:

Please also check [Django's documentation](#) for parameters which configure Django itself.

4.30.1 AKISMET_API_KEY

Weblate can use Akismet to check incoming anonymous suggestions for spam. Visit akismet.com to purchase an API key and associate it with a site.

4.30.2 ANONYMOUS_USER_NAME

User name of user for defining privileges of not logged in user.

See also:

Access control

4.30.3 AUDITLOG_EXPIRY

New in version 3.6.

How long (in days) Weblate should keep audit log containing information about account activity.

Defaults to 180 days.

4.30.4 AUTH_LOCK_ATTEMPTS

New in version 2.14.

Maximum number of failed authentication attempts before rate limiting is applied.

This is currently applied in the following locations:

- On login, the account password is reset. User will not be able to log in after that using password until he asks for password reset.
- On password reset, the reset mails are no longer sent. This avoids spamming user with too many password reset attempts.

Defaults to 10.

See also:

Rate limiting,

4.30.5 AUTO_UPDATE

New in version 3.2.

Automatically update all repositories on daily basis. This can be useful if you do not use *Notification hooks* to update Weblate repositories automatically.

Note: This requires *Background tasks using Celery* working and you will have to restart celery for this setting to take effect.

4.30.6 AVATAR_URL_PREFIX

Prefix for constructing avatar URLs. The URL will be constructed like: `${AVATAR_URL_PREFIX}/avatar/${MAIL_HASH}?${PARAMS}`. Following services are known to work:

Gravatar (default), see <https://gravatar.com/> `AVATAR_URL_PREFIX = 'https://www.gravatar.com/'`

Libravatar, see <https://www.libravatar.org/> `AVATAR_URL_PREFIX = 'https://seccdn.libravatar.org/'`

See also:

Avatar caching, `ENABLE_AVATARS`, *Avatars*

4.30.7 RATELIMIT_ATTEMPTS

New in version 3.2.

Maximum number of authentication attempts before rate limiting applies.

Defaults to 5.

See also:

Rate limiting, `RATELIMIT_WINDOW`, `RATELIMIT_LOCKOUT`

4.30.8 RATELIMIT_WINDOW

New in version 3.2.

Length of authentication window for rate limiting in seconds.

Defaults to 300 (5 minutes).

See also:

Rate limiting, `RATELIMIT_ATTEMPTS`, `RATELIMIT_LOCKOUT`

4.30.9 RATELIMIT_LOCKOUT

New in version 3.2.

Length of authentication lockout window after rate limit is applied.

Defaults to 600 (10 minutes).

See also:

Rate limiting, `RATELIMIT_ATTEMPTS`, `RATELIMIT_WINDOW`

4.30.10 AUTH_TOKEN_VALID

New in version 2.14.

Validity of token in activation and password reset mails in seconds.

Defaults to 3600 (1 hour).

4.30.11 AUTH_PASSWORD_DAYS

New in version 2.15.

Define (in days) how long in past Weblate should reject reusing same password.

Note: Password changes done prior to Weblate 2.15 will not be accounted for this policy, it is valid only

Defaults to 180 days.

4.30.12 AUTOFIX_LIST

List of automatic fixups to apply when saving the message.

You need to provide a fully-qualified path to the Python class implementing the autofixer interface.

Available fixes:

`weblate.trans.autofixes.whitespace.SameBookendingWhitespace` Fixes up whitespace in beginning and end of the string to match source.

`weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis` Replaces trailing dots with ellipsis if source string has it.

`weblate.trans.autofixes.chars.RemoveZeroSpace` Removes zero width space char if source does not contain it.

`weblate.trans.autofixes.chars.RemoveControlCharS` Removes control characters if source does not contain it.

For example you can enable only few of them:

```
AUTOFIX_LIST = (  
    'weblate.trans.autofixes.whitespace.SameBookendingWhitespace',  
    'weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis',  
)
```

See also:

Automatic fixups, Custom automatic fixups

4.30.13 BASE_DIR

Base directory where Weblate sources are located. This is used to derive several other paths by default:

- `DATA_DIR`

Default value: Top level directory of Weblate sources.

4.30.14 CHECK_LIST

List of quality checks to perform on translation.

You need to provide a fully-qualified path to the Python class implementing the check interface.

Some of the checks are not useful for all projects, so you are welcome to adjust the list of checks to be performed on your installation.

By default all built in quality checks (see [Quality checks](#)) are enabled, you can use this setting to change this. Also the [Sample configuration](#) comes with this setting commented out to use default value. This enables you to get new checks automatically enabled on upgrade.

You can disable all checks:

```
CHECK_LIST = ()
```

You can enable only few of them:

```
CHECK_LIST = (  
    'weblate.checks.chars.BeginNewlineCheck',  
    'weblate.checks.chars.EndNewlineCheck',  
    'weblate.checks.chars.MaxLengthCheck',  
)
```

Note: Once you change this setting the existing checks will still be stored in the database, only newly changed translations will be affected by the change. To apply the change to the stored translations, you need to run [updatechecks](#).

See also:

[Quality checks](#), [Customizing behavior](#)

4.30.15 COMMENT_CLEANUP_DAYS

New in version 3.6.

Automatically delete comments after given number of days. Defaults to `None` what means no deletion at all.

4.30.16 COMMIT_PENDING_HOURS

New in version 2.10.

Default interval for committing pending changes using [commit_pending](#).

See also:

[Running maintenance tasks](#), [commit_pending](#)

4.30.17 DATA_DIR

Directory where Weblate stores all data. This consists of VCS repositories, fulltext index and various configuration files for external tools.

The following subdirectories usually exist:

home Home directory used for invoking scripts.

ssh SSH keys and configuration.

static Default location for Django static files, specified by `STATIC_ROOT`.

media Default location for Django media files, specified by `MEDIA_ROOT`.

memory Translation memory data using Whoosh engine (see *Translation Memory*).

vcs Version control repositories.

whoosh Fulltext search index using Whoosh engine.

backups Dump of data in daily backups, see *Dumped data for backups*.

Note: This directory has to be writable by Weblate. If you are running Weblate as uwsgi this means that it should be writable by the `www-data` user.

The easiest way to achieve is to make the user own the directory:

```
sudo chown www-data:www-data -R $DATA_DIR
```

Defaults to `$BASE_DIR/data`.

See also:

BASE_DIR, *Backing up and moving Weblate*

4.30.18 DEFAULT_ACCESS_CONTROL

New in version 3.3.

Choose default access control when creating new project, possible values are currently:

0 *Public*

1 *Protected*

100 *Private*

200 *Custom*

Use *Custom* if you are going to manage ACL manually and do not want to rely on Weblate internal management.

See also:

Per project access control, *Access control*

4.30.19 DEFAULT_COMMITER_EMAIL

New in version 2.4.

Default committer email when creating translation component (see *Component configuration*), defaults to `noreply@weblate.org`.

See also:

DEFAULT_COMMITER_NAME, *Component configuration*

4.30.20 DEFAULT_COMMITER_NAME

New in version 2.4.

Default committer name when creating translation component (see *Component configuration*), defaults to Weblate.

See also:

DEFAULT_COMMITER_EMAIL, *Component configuration*

4.30.21 DEFAULT_MERGE_STYLE

New in version 3.4.

Default merge style for new components (see *Component configuration*), choose one of:

- *rebase* - default
- *merge*

4.30.22 DEFAULT_TRANSLATION_PROPAGATION

New in version 2.5.

Default setting for translation propagation (see *Component configuration*), defaults to `True`.

See also:

Component configuration

4.30.23 DEFAULT_PULL_MESSAGE

Default pull request title, defaults to 'Update from Weblate'.

4.30.24 ENABLE_AVATARS

Whether to enable Gravatar based avatars for users. By default this is enabled.

The avatars are fetched and cached on the server, so there is no risk in leaking private information or slowing down the user experiences with enabling this.

See also:

Avatar caching, *AVATAR_URL_PREFIX*, *Avatars*

4.30.25 ENABLE_HOOKS

Whether to enable anonymous remote hooks.

See also:

Notification hooks

4.30.26 ENABLE_HTTPS

Whether to send links to Weblate as https or http. This setting affects sent mails and generated absolute URLs.

See also:

Set correct sitename

4.30.27 ENABLE_SHARING

Whether to show links to share translation progress on social networks.

4.30.28 GITHUB_USERNAME

GitHub username that will be used to send pull requests for translation updates.

See also:

Pushing changes to GitHub as pull request, Setting up hub

4.30.29 GOOGLE_ANALYTICS_ID

Google Analytics ID to enable monitoring of Weblate using Google Analytics.

4.30.30 HIDE_REPO_CREDENTIALS

Hide repository credentials in the web interface. In case you have repository URL with user and password, Weblate will hide it when showing it to the users.

For example instead of `https://user:password@git.example.com/repo.git` it will show just `https://git.example.com/repo.git`. It tries to cleanup VCS error messages as well in similar manner.

This is enabled by default.

4.30.31 IP_BEHIND_REVERSE_PROXY

New in version 2.14.

Indicates whether Weblate is running behind a reverse proxy.

If set to True, Weblate gets IP address from header defined by `IP_BEHIND_REVERSE_PROXY`. Ensure that you are actually using reverse proxy and that it sets this header, otherwise users will be able to fake the IP address.

Defaults to False.

See also:

Rate limiting, IP address for rate limiting

4.30.32 IP_PROXY_HEADER

New in version 2.14.

Indicates from which header Weblate should obtain the IP address when `IP_BEHIND_REVERSE_PROXY` is enabled.

Defaults to `HTTP_X_FORWARDED_FOR`.

See also:

Rate limiting, IP address for rate limiting

4.30.33 IP_PROXY_OFFSET

New in version 2.14.

Indicates which part of `IP_BEHIND_REVERSE_PROXY` is used as client IP address.

Depending on your setup, this header might consist of several IP addresses, (for example `X-Forwarded-For: a, b, client-ip`) and you can configure here which address from the header is client IP address.

Warning: Setting this affects security of your installation, you should only configure to use trusted proxies for determining IP address.

Defaults to 0.

See also:

Rate limiting, IP address for rate limiting

4.30.34 LEGAL_URL

New in version 3.5.

URL where your Weblate instance shows it's legal documents. This is useful if you host your legal documents outside Weblate for embedding inside Weblate please see [Legal](#).

4.30.35 LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH

By default the length of a given translation is limited to the length of the source string * 10 characters. Set this option to False to allow longer translations (up to 10.000 characters) irrespective of the source length.

Defaults to True.

4.30.36 LOGIN_REQUIRED_URLS

List of URLs which require login (besides standard rules built into Weblate). This allows you to password protect whole installation using:

```
LOGIN_REQUIRED_URLS = (
    r'/(.*)$',
)
```

4.30.37 LOGIN_REQUIRED_URLS_EXCEPTIONS

List of exceptions for `LOGIN_REQUIRED_URLS`. If you don't specify this list, the default value will be used, which allows users to access the login page.

Some of exceptions you might want to include:

```
LOGIN_REQUIRED_URLS_EXCEPTIONS = (  
    r'/accounts/(.*)$', # Required for login  
    r'/static/(.*)$',   # Required for development mode  
    r'/widgets/(.*)$',  # Allowing public access to widgets  
    r'/data/(.*)$',     # Allowing public access to data exports  
    r'/hooks/(.*)$',    # Allowing public access to notification hooks  
    r'/api/(.*)$',      # Allowing access to API  
    r'/js/i18n/$',      # JavaScript localization  
)
```

4.30.38 MT_SERVICES

Changed in version 3.0: The setting was renamed from `MACHINE_TRANSLATION_SERVICES` to `MT_SERVICES` to be consistent with other machine translation settings.

List of enabled machine translation services to use.

Note: Many of services need additional configuration like API keys, please check their documentation for more details.

```
MT_SERVICES = (  
    'weblate.machinery.apertium.ApertiumAPYTranslation',  
    'weblate.machinery.deepl.DeepLTranslation',  
    'weblate.machinery.glosbe.GlosbeTranslation',  
    'weblate.machinery.google.GoogleTranslation',  
    'weblate.machinery.microsoft.MicrosoftCognitiveTranslation',  
    'weblate.machinery.microsoftterminology.MicrosoftTerminologyService',  
    'weblate.machinery.mymemory.MyMemoryTranslation',  
    'weblate.machinery.tmserver.AmagamaTranslation',  
    'weblate.machinery.tmserver.TMServerTranslation',  
    'weblate.machinery.yandex.YandexTranslation',  
    'weblate.machinery.weblatetm.WeblateTranslation',  
    'weblate.machinery.saptranslationhub.SAPTranslationHub',  
    'weblate.memory.machine.WeblateMemory',  
)
```

See also:

Machine translation, Machine translation

4.30.39 MT_APERTIUM_APY

URL of the Apertium APy server, see <http://wiki.apertium.org/wiki/Apertium-apy>

See also:

Apertium, Machine translation, Machine translation

4.30.40 MT_AWS_ACCESS_KEY_ID

Access key ID for Amazon Translate.

See also:

AWS, Machine translation, Machine translation

4.30.41 MT_AWS_SECRET_ACCESS_KEY

API secret key for Amazon Translate.

See also:

AWS, Machine translation, Machine translation

4.30.42 MT_AWS_REGION

Region name to use for Amazon Translate.

See also:

AWS, Machine translation, Machine translation

4.30.43 MT_Baidu_ID

Client ID for Baidu Zhiyun API, you can register at <https://api.fanyi.baidu.com/api/trans/product/index>

See also:

Baidu API machine translation, Machine translation, Machine translation

4.30.44 MT_Baidu_SECRET

Client secret for Baidu Zhiyun API, you can register at <https://api.fanyi.baidu.com/api/trans/product/index>

See also:

Baidu API machine translation, Machine translation, Machine translation

4.30.45 MT_DEEPL_KEY

API key for DeepL API, you can register at <https://www.deepl.com/pro.html>.

See also:

DeepL, Machine translation, Machine translation

4.30.46 MT_GOOGLE_KEY

API key for Google Translate API, you can register at <https://cloud.google.com/translate/docs>

See also:

Google Translate, Machine translation, Machine translation

4.30.47 MT_MICROSOFT_COGNITIVE_KEY

Client key for Microsoft Cognitive Services Translator API.

See also:

Microsoft Cognitive Services Translator, Machine translation, Machine translation, Cognitive Services - Text Translation API, Microsoft Azure Portal

4.30.48 MT_MYMEMORY_EMAIL

MyMemory identification email, you can get 1000 requests per day with this.

See also:

MyMemory, Machine translation, Machine translation, MyMemory: API technical specifications

4.30.49 MT_MYMEMORY_KEY

MyMemory access key for private translation memory, use together with *MT_MYMEMORY_USER*.

See also:

MyMemory, Machine translation, Machine translation, MyMemory: API key generator

4.30.50 MT_MYMEMORY_USER

MyMemory user id for private translation memory, use together with *MT_MYMEMORY_KEY*.

See also:

MyMemory, Machine translation, Machine translation, MyMemory: API key generator

4.30.51 MT_NETEASE_KEY

App key for Netease Sight API, you can register at <https://sight.netease.com/>

See also:

Netease Sight API machine translation, Machine translation, Machine translation

4.30.52 MT_NETEASE_SECRET

App secret for Netease Sight API, you can register at <https://sight.netease.com/>

See also:

Netease Sight API machine translation, Machine translation, Machine translation

4.30.53 MT_TMSERVER

URL where tmserver is running.

See also:

tmserver, Machine translation, Machine translation, tmserver

4.30.54 MT_YANDEX_KEY

API key for Yandex Translate API, you can register at <https://tech.yandex.com/translate/>

See also:

Yandex Translate, Machine translation, Machine translation

4.30.55 MT_YOUDAO_ID

Client ID for Youdao Zhiyun API, you can register at <https://ai.youdao.com/product-fanyi.s>

See also:

Youdao Zhiyun API machine translation, Machine translation, Machine translation

4.30.56 MT_YOUDAO_SECRET

Client secret for Youdao Zhiyun API, you can register at <https://ai.youdao.com/product-fanyi.s>

See also:

Youdao Zhiyun API machine translation, Machine translation, Machine translation

4.30.57 MT_SAP_BASE_URL

API URL to the SAP Translation Hub service.

See also:

SAP Translation Hub, Machine translation, Machine translation

4.30.58 MT_SAP_SANDBOX_APIKEY

API key for sandbox API usage

See also:

SAP Translation Hub, Machine translation, Machine translation

4.30.59 MT_SAP_USERNAME

Your SAP username

See also:

SAP Translation Hub, Machine translation, Machine translation

4.30.60 MT_SAP_PASSWORD

Your SAP password

See also:

SAP Translation Hub, Machine translation, Machine translation

4.30.61 MT_SAP_USE_MT

Should the machine translation service also be used? (in addition to the term database). Possible values: True / False

See also:

SAP Translation Hub, Machine translation, Machine translation

4.30.62 NEARBY_MESSAGES

How many messages around current one to show during translating.

4.30.63 PIWIK_SITE_ID

ID of a site in Matomo you want to track.

See also:

PIWIK_URL

4.30.64 PIWIK_URL

URL of a Matomo installation you want to use to track Weblate users. For more information about Matomo see <https://matomo.org/>.

See also:

PIWIK_SITE_ID

4.30.65 REGISTRATION_CAPTCHA

A boolean (either `True` or `False`) indicating whether registration of new accounts is protected by captcha. This setting is optional, and a default of `True` will be assumed if it is not supplied.

If enabled the captcha is added to all pages where users enter email address:

- New account registration.
- Password recovery.
- Adding email to an account.
- Contact form for users who are not logged in.

4.30.66 REGISTRATION_EMAIL_MATCH

New in version 2.17.

Allows you to filter email addresses which can register.

Defaults to `.*` which allows any address to register.

You can use it to restrict registration to a single email domain:

```
REGISTRATION_EMAIL_MATCH = r'^.*@weblate\.org$'
```


4.30.67 REGISTRATION_OPEN

A boolean (either `True` or `False`) indicating whether registration of new accounts is currently permitted. This setting is optional, and a default of `True` will be assumed if it is not supplied.

4.30.68 SIMPLIFY_LANGUAGES

Use simple language codes for default language/country combinations. For example `fr_FR` translation will use `fr` language code. This is usually desired behavior as it simplifies listing of the languages for these default combinations.

Disable this if you are having different translations for both variants.

4.30.69 SITE_TITLE

Site title to be used in website and emails as well.

4.30.70 SPECIAL_CHARS

Additional chars to show in the visual keyboard, see *Visual keyboard*.

The default value is:

```
SPECIAL_CHARS = ('\t', '\n', '...')
```

4.30.71 STATUS_URL

URL where your Weblate instance reports it's status.

4.30.72 SUGGESTION_CLEANUP_DAYS

New in version 3.2.1.

Automatically delete suggestions after given number of days. Defaults to `None` what means no deletion at all.

4.30.73 URL_PREFIX

This settings allows you to run Weblate under some path (otherwise it relies on being executed from webserver root). To use this setting, you also need to configure your server to strip this prefix. For example with WSGI, this can be achieved by setting `WSGIScriptAlias`.

Note: This setting does not work with Django's builtin server, you would have to adjust `urls.py` to contain this prefix.

4.30.74 WEBLATE_ADDONS

List of addons available for use. To use them, they have to be enabled for given translation component. By default this includes all built in addons, when extending the list you will probably want to keep existing ones enabled, for example:

```
WEBLATE_ADDONS = (  
    # Built in addons  
    'weblate.addons.gettext.GenerateMoAddon',  
    'weblate.addons.gettext.UpdateLinguasAddon',  
    'weblate.addons.gettext.UpdateConfigureAddon',  
    'weblate.addons.gettext.MsgmergeAddon',  
    'weblate.addons.gettext.GettextCustomizeAddon',  
    'weblate.addons.gettext.GettextAuthorComments',  
    'weblate.addons.cleanup.CleanupAddon',  
    'weblate.addons.consistency.LanguaugeConsistencyAddon',  
    'weblate.addons.discovery.DiscoveryAddon',  
    'weblate.addons.flags.SourceEditAddon',  
    'weblate.addons.flags.TargetEditAddon',  
    'weblate.addons.flags.SameEditAddon',  
    'weblate.addons.generate.GenerateFileAddon',  
    'weblate.addons.json.JSONCustomizeAddon',  
    'weblate.addons.properties.PropertiesSortAddon',  
    'weblate.addons.git.GitSquashAddon',  
    'weblate.addons.removal.RemoveComments',  
    'weblate.addons.removal.RemoveSuggestions',  
  
    # Addon you want to include  
    'weblate.addons.example.ExampleAddon',  
)
```

See also:

Addons

4.30.75 WEBLATE_FORMATS

New in version 3.0.

List of file formats available for use, you can usually keep this on default value.

See also:

Supported file formats

4.30.76 WEBLATE_GPG_IDENTITY

New in version 3.1.

Identity which should be used by Weblate to sign Git commits, for example:

```
WEBLATE_GPG_IDENTITY = 'Weblate <weblate@example.com>'
```

Warning: If you are going to change value of setting, it is advisable to clean the cache as the key information is cached for seven days. This is not necessary for initial setup as nothing is cached if this feature is not configured.

See also:*Signing Git commits by GnuPG*

4.31 Sample configuration

The following example is shipped as `weblate/settings_example.py` with Weblate:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2019 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <https://www.gnu.org/licenses/>.
#

from __future__ import unicode_literals

import os
import platform
from logging.handlers import SysLogHandler

#
# Django settings for Weblate project.
#

DEBUG = True

ADMINS = (
    # ('Your Name', 'your_email@example.com'),
)

MANAGERS = ADMINS

DATABASES = {
    'default': {
        # Use 'postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'ENGINE': 'django.db.backends.sqlite3',
        # Database name or path to database file if using sqlite3.
        'NAME': 'weblate.db',
        # Database user, not used with sqlite3.
        'USER': 'weblate',
        # Database password, not used with sqlite3.
        'PASSWORD': 'weblate',
```

(continues on next page)

(continued from previous page)

```

# Set to empty string for localhost. Not used with sqlite3.
'HOST': '127.0.0.1',
# Set to empty string for default. Not used with sqlite3.
'PORT': '',
# Customizations for databases
'OPTIONS': {
    # In case of using an older MySQL server,
    # which has MyISAM as a default storage
    # 'init_command': 'SET storage_engine=INNODB',
    # Uncomment for MySQL older than 5.7:
    # 'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
    # Set emoji capable charset for MySQL:
    # 'charset': 'utf8mb4',
},
}
}

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Data directory
DATA_DIR = os.path.join(BASE_DIR, 'data')

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# In a Windows environment this must be set to your system time zone.
TIME_ZONE = 'UTC'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-us'

LANGUAGES = (
    ('ar', ''),
    ('az', 'Azərbaycan'),
    ('be', ''),
    ('be@latin', 'Беларуская'),
    ('bg', ''),
    ('br', 'Brezhoneg'),
    ('ca', 'Català'),
    ('cs', 'Čeština'),
    ('da', 'Dansk'),
    ('de', 'Deutsch'),
    ('en', 'English'),
    ('en-gb', 'English (United Kingdom)'),
    ('el', 'Ελληνικά'),
    ('es', 'Español'),
    ('fi', 'Suomi'),
    ('fr', 'Français'),
    ('fy', 'Frysk'),
    ('gl', 'Galego'),
    ('he', ''),
    ('hu', 'Magyar'),
    ('id', 'Indonesia'),
    ('it', 'Italiano'),
    ('ja', ''),
    ('kk', ' '),

```

(continues on next page)

(continued from previous page)

```

('ko', ''),
('ksh', 'Kölsch'),
('nb', 'Norsk bokmål'),
('nl', 'Nederlands'),
('pl', 'Polski'),
('pt', 'Português'),
('pt-br', 'Português brasileiro'),
('ru', ''),
('sk', 'Slovenčina'),
('sl', 'Slovenščina'),
('sr', ''),
('sv', 'Svenska'),
('tr', 'Türkçe'),
('uk', ''),
('zh-hans', ''),
('zh-hant', ''),
)

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as not
# to load the internationalization machinery.
USE_I18N = True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale.
USE_L10N = True

# If you set this to False, Django will not use timezone-aware datetimes.
USE_TZ = True

# URL prefix to use, please see documentation for more details
URL_PREFIX = ''

# Absolute filesystem path to the directory that will hold user-uploaded files.
# Example: "/home/media/media.lawrence.com/media/"
MEDIA_ROOT = os.path.join(DATA_DIR, 'media')

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash.
# Examples: "http://media.lawrence.com/media/", "http://example.com/media/"
MEDIA_URL = '{0}/media/'.format(URL_PREFIX)

# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT = os.path.join(DATA_DIR, 'static')

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL = '{0}/static/'.format(URL_PREFIX)

# Additional locations of static files
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or "C:/www/django/static".
    # Always use forward slashes, even on Windows.

```

(continues on next page)

(continued from previous page)

```

    # Don't forget to use absolute paths, not relative paths.
)

# List of finder classes that know how to find static files in
# various locations.
STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
    'compressor.finders.CompressorFinder',
)

# Make this unique, and don't share it with anybody.
# You can generate it using weblate/examples/generate-secret-key
SECRET_KEY = 'jm8fqjlg+5!#xu%e-oh#7!$aa7!6avf7ud*_v=chdrb9qdc06(' # noqa

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'weblate', 'templates'),
        ],
        'OPTIONS': {
            'context_processors': [
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.debug',
                'django.template.context_processors.i18n',
                'django.template.context_processors.request',
                'django.template.context_processors.csrf',
                'django.contrib.messages.context_processors.messages',
                'weblate.trans.context_processors.weblate_context',
            ],
            'loaders': [
                ('django.template.loaders.cached.Loader', [
                    'django.template.loaders.filesystem.Loader',
                    'django.template.loaders.app_directories.Loader',
                ]),
            ],
        },
    },
]

# GitHub username for sending pull requests.
# Please see the documentation for more details.
GITHUB_USERNAME = None

# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.email.EmailAuth',
    # 'social_core.backends.google.GoogleOAuth2',
    # 'social_core.backends.github.GithubOAuth2',
    # 'social_core.backends.bitbucket.BitbucketOAuth',
    # 'social_core.backends.suse.OpenSUSEOpenId',
    # 'social_core.backends.ubuntu.UbuntuOpenId',
    # 'social_core.backends.fedora.FedoraOpenId',
    # 'social_core.backends.facebook.FacebookOAuth2',
    'weblate.accounts.auth.WeblateUserBackend',
)

```

(continues on next page)

(continued from previous page)

```

)

# Custom user model
AUTH_USER_MODEL = 'weblate_auth.User'

# Social auth backends setup
SOCIAL_AUTH_GITHUB_KEY = ''
SOCIAL_AUTH_GITHUB_SECRET = ''
SOCIAL_AUTH_GITHUB_SCOPE = ['user:email']

SOCIAL_AUTH_BITBUCKET_KEY = ''
SOCIAL_AUTH_BITBUCKET_SECRET = ''
SOCIAL_AUTH_BITBUCKET_VERIFIED_EMAILS_ONLY = True

SOCIAL_AUTH_FACEBOOK_KEY = ''
SOCIAL_AUTH_FACEBOOK_SECRET = ''
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email', 'public_profile']
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {'fields': 'id,name,email'}
SOCIAL_AUTH_FACEBOOK_API_VERSION = '3.1'

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ''
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ''

# Social auth settings
SOCIAL_AUTH_PIPELINE = (
    'social_core.pipeline.social_auth.social_details',
    'social_core.pipeline.social_auth.social_uid',
    'social_core.pipeline.social_auth.auth_allowed',
    'social_core.pipeline.social_auth.social_user',
    'weblate.accounts.pipeline.store_params',
    'weblate.accounts.pipeline.verify_open',
    'social_core.pipeline.user.get_username',
    'weblate.accounts.pipeline.require_email',
    'social_core.pipeline.mail.mail_validation',
    'weblate.accounts.pipeline.revoke_mail_code',
    'weblate.accounts.pipeline.ensure_valid',
    'weblate.accounts.pipeline.remove_account',
    'social_core.pipeline.social_auth.associate_by_email',
    'weblate.accounts.pipeline.reauthenticate',
    'weblate.accounts.pipeline.verify_username',
    'social_core.pipeline.user.create_user',
    'social_core.pipeline.social_auth.associate_user',
    'social_core.pipeline.social_auth.load_extra_data',
    'weblate.accounts.pipeline.cleanup_next',
    'weblate.accounts.pipeline.user_full_name',
    'weblate.accounts.pipeline.store_email',
    'weblate.accounts.pipeline.notify_connect',
    'weblate.accounts.pipeline.password_reset',
)
SOCIAL_AUTH_DISCONNECT_PIPELINE = (
    'social_core.pipeline.disconnect.allowed_to_disconnect',
    'social_core.pipeline.disconnect.get_entries',
    'social_core.pipeline.disconnect.revoke_tokens',
    'weblate.accounts.pipeline.cycle_session',
    'weblate.accounts.pipeline.adjust_primary_mail',
    'weblate.accounts.pipeline.notify_disconnect',
    'social_core.pipeline.disconnect.disconnect',
)

```

(continues on next page)

(continued from previous page)

```

        'weblate.accounts.pipeline.cleanup_next',
    )

    # Custom authentication strategy
    SOCIAL_AUTH_STRATEGY = 'weblate.accounts.strategy.WeblateStrategy'

    # Raise exceptions so that we can handle them later
    SOCIAL_AUTH_RAISE_EXCEPTIONS = True

    SOCIAL_AUTH_EMAIL_VALIDATION_FUNCTION = \
        'weblate.accounts.pipeline.send_validation'
    SOCIAL_AUTH_EMAIL_VALIDATION_URL = \
        '{0}/accounts/email-sent/'.format(URL_PREFIX)
    SOCIAL_AUTH_LOGIN_ERROR_URL = \
        '{0}/accounts/login/'.format(URL_PREFIX)
    SOCIAL_AUTH_EMAIL_FORM_URL = \
        '{0}/accounts/email/'.format(URL_PREFIX)
    SOCIAL_AUTH_NEW_ASSOCIATION_REDIRECT_URL = \
        '{0}/accounts/profile/#account'.format(URL_PREFIX)
    SOCIAL_AUTH_PROTECTED_USER_FIELDS = ('email',)
    SOCIAL_AUTH_SLUGIFY_USERNAMES = True
    SOCIAL_AUTH_SLUGIFY_FUNCTION = 'weblate.accounts.pipeline.slugify_username'

    # Password validation configuration
    AUTH_PASSWORD_VALIDATORS = [
        {
            'NAME': 'django.contrib.auth.password_validation.
→UserAttributeSimilarityValidator', # noqa: E501, pylint: disable=line-too-long
        },
        {
            'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
            'OPTIONS': {
                'min_length': 6,
            }
        },
        {
            'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
            'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
        },
        {
            'NAME': 'weblate.accounts.password_validation.CharsPasswordValidator',
        },
        {
            'NAME': 'weblate.accounts.password_validation.PastPasswordsValidator',
        },
        # Optional password strength validation by django-zxcvbn-password
        # {
        #     'NAME': 'zxcvbn_password.ZXCVBNValidator',
        #     'OPTIONS': {
        #         'min_score': 3,
        #         'user_attributes': ('username', 'email', 'full_name')
        #     }
        # },
    ]

```

(continues on next page)

(continued from previous page)

```

# Allow new user registrations
REGISTRATION_OPEN = True

# Middleware
MIDDLEWARE = [
    'weblate.middleware.ProxyMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'weblate.accounts.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'social_django.middleware.SocialAuthExceptionMiddleware',
    'weblate.accounts.middleware.RequireLoginMiddleware',
    'weblate.middleware.SecurityMiddleware',
]

ROOT_URLCONF = 'weblate.urls'

# Django and Weblate apps
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin.apps.SimpleAdminConfig',
    'django.contrib.admindocs',
    'django.contrib.sitemaps',
    'social_django',
    'crispy_forms',
    'compressor',
    'rest_framework',
    'rest_framework.authtoken',
    'weblate.addons',
    'weblate.auth',
    'weblate.checks',
    'weblate.formats',
    'weblate.machinery',
    'weblate.trans',
    'weblate.lang',
    'weblate.langdata',
    'weblate.memory',
    'weblate.screenshots',
    'weblate.fonts',
    'weblate.accounts',
    'weblate.utils',
    'weblate.vcs',
    'weblate.wladmin',
    'weblate',

    # Optional: Git exporter
    # 'weblate.gitexport',
)

```

(continues on next page)

(continued from previous page)

```

# Path to locales
LOCALE_PATHS = (os.path.join(BASE_DIR, 'weblate', 'locale'), )

# Custom exception reporter to include some details
DEFAULT_EXCEPTION_REPORTER_FILTER = \
    'weblate.trans.debug.WeblateExceptionReporterFilter'

# Default logging of Weblate messages
# - to syslog in production (if available)
# - otherwise to console
# - you can also choose 'logfile' to log into separate file
#   after configuring it below

# Detect if we can connect to syslog
HAVE_SYSLOG = False
if platform.system() != 'Windows':
    try:
        handler = SysLogHandler(
            address='/dev/log', facility=SysLogHandler.LOG_LOCAL2
        )
        handler.close()
        HAVE_SYSLOG = True
    except IOError:
        HAVE_SYSLOG = False

if DEBUG or not HAVE_SYSLOG:
    DEFAULT_LOG = 'console'
else:
    DEFAULT_LOG = 'syslog'

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error when DEBUG=False.
# See http://docs.djangoproject.com/en/stable/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'formatters': {
        'syslog': {
            'format': 'weblate[%(process)d]: %(levelname)s %(message)s'
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
        'logfile': {
            'format': '%(asctime)s %(levelname)s %(message)s'
        },
        'django.server': {
            '()': 'django.utils.log.ServerFormatter',
            'format': '[%(server_time)s] %(message)s',

```

(continues on next page)

(continued from previous page)

```

    }
},
'handlers': {
    'mail_admins': {
        'level': 'ERROR',
        'filters': ['require_debug_false'],
        'class': 'django.utils.log.AdminEmailHandler',
        'include_html': True,
    },
    'console': {
        'level': 'DEBUG',
        'class': 'logging.StreamHandler',
        'formatter': 'simple'
    },
    'django.server': {
        'level': 'INFO',
        'class': 'logging.StreamHandler',
        'formatter': 'django.server',
    },
    'syslog': {
        'level': 'DEBUG',
        'class': 'logging.handlers.SysLogHandler',
        'formatter': 'syslog',
        'address': '/dev/log',
        'facility': SysLogHandler.LOG_LOCAL2,
    },
    # Logging to a file
    # 'logfile': {
    #     'level': 'DEBUG',
    #     'class': 'logging.handlers.RotatingFileHandler',
    #     'filename': "/var/log/weblate/weblate.log",
    #     'maxBytes': 100000,
    #     'backupCount': 3,
    #     'formatter': 'logfile',
    # },
},
'loggers': {
    'django.request': {
        'handlers': ['mail_admins', DEFAULT_LOG],
        'level': 'ERROR',
        'propagate': True,
    },
    'django.server': {
        'handlers': ['django.server'],
        'level': 'INFO',
        'propagate': False,
    },
    # Logging database queries
    # 'django.db.backends': {
    #     'handlers': [DEFAULT_LOG],
    #     'level': 'DEBUG',
    # },
    'weblate': {
        'handlers': [DEFAULT_LOG],
        'level': 'DEBUG',
    },
    # Logging search operations

```

(continues on next page)

(continued from previous page)

```

        'weblate.search': {
            'handlers': [DEFAULT_LOG],
            'level': 'INFO',
        },
        # Logging VCS operations
        'weblate.vcs': {
            'handlers': [DEFAULT_LOG],
            'level': 'WARNING',
        },
        # Python Social Auth logging
        # 'social': {
        #     'handlers': [DEFAULT_LOG],
        #     'level': 'DEBUG',
        # },
    }
}

# Logging of management commands to console
if (os.environ.get('DJANGO_IS_MANAGEMENT_COMMAND', False)
    and 'console' not in LOGGING['loggers']['weblate']['handlers']):
    LOGGING['loggers']['weblate']['handlers'].append('console')

# Remove syslog setup if it's not present
if not HAVE_SYSLOG:
    del LOGGING['handlers']['syslog']

# List of machine translations
# MT_SERVICES = (
#     'weblate.machinery.apertium.ApertiumAPYTranslation',
#     'weblate.machinery.baidu.BaiduTranslation',
#     'weblate.machinery.deepl.DeepLTranslation',
#     'weblate.machinery.glosbe.GlosbeTranslation',
#     'weblate.machinery.google.GoogleTranslation',
#     'weblate.machinery.microsoft.MicrosoftCognitiveTranslation',
#     'weblate.machinery.microsoftterminology.MicrosoftTerminologyService',
#     'weblate.machinery.mymemory.MyMemoryTranslation',
#     'weblate.machinery.netease.NeteaseSightTranslation',
#     'weblate.machinery.tmserver.AmagamaTranslation',
#     'weblate.machinery.tmserver.TMServerTranslation',
#     'weblate.machinery.yandex.YandexTranslation',
#     'weblate.machinery.weblatetm.WeblateTranslation',
#     'weblate.machinery.saptranslationhub.SAPTranslationHub',
#     'weblate.machinery.youdao.YoudaoTranslation',
#     'weblate.memory.machine.WeblateMemory',
# )

# Machine translation API keys

# URL of the Apertium APY server
MT_APERTIUM_APY = None

# DeepL API key
MT_DEEPL_KEY = None

# Microsoft Cognitive Services Translator API, register at
# https://portal.azure.com/
MT_MICROSOFT_COGNITIVE_KEY = None

```

(continues on next page)

(continued from previous page)

```

# MyMemory identification email, see
# https://mymemory.translated.net/doc/spec.php
MT_MYMEMORY_EMAIL = None

# Optional MyMemory credentials to access private translation memory
MT_MYMEMORY_USER = None
MT_MYMEMORY_KEY = None

# Google API key for Google Translate API
MT_GOOGLE_KEY = None

# Baidu app key and secret
MT_BAIDU_ID = None
MT_BAIDU_SECRET = None

# Youdao Zhiyun app key and secret
MT_YOUDAO_ID = None
MT_YOUDAO_SECRET = None

# Netease Sight (Jianwai) app key and secret
MT_NETEASE_KEY = None
MT_NETEASE_SECRET = None

# API key for Yandex Translate API
MT_YANDEX_KEY = None

# tmserver URL
MT_TMSERVER = None

# SAP Translation Hub
MT_SAP_BASE_URL = None
MT_SAP_SANDBOX_APIKEY = None
MT_SAP_USERNAME = None
MT_SAP_PASSWORD = None
MT_SAP_USE_MT = True

# Title of site to use
SITE_TITLE = 'Weblate'

# Whether site uses https
ENABLE_HTTPS = False

# Use HTTPS when creating redirect URLs for social authentication, see
# documentation for more details:
# https://python-social-auth-docs.readthedocs.io/en/latest/configuration/settings.html
→ #processing-redirects-and-urlopen
SOCIAL_AUTH_REDIRECT_IS_HTTPS = ENABLE_HTTPS

# Make CSRF cookie HttpOnly, see documentation for more details:
# https://docs.djangoproject.com/en/1.11/ref/settings/#csrf-cookie-httponly
CSRF_COOKIE_HTTPONLY = True
CSRF_COOKIE_SECURE = ENABLE_HTTPS
# Store CSRF token in session
CSRF_USE_SESSIONS = True
# Customize CSRF failure view
CSRF_FAILURE_VIEW = 'weblate.trans.views.error.csrf_failure'

```

(continues on next page)

(continued from previous page)

```
SESSION_COOKIE_SECURE = ENABLE_HTTPS
# SSL redirect
SECURE_SSL_REDIRECT = ENABLE_HTTPS
# SSL redirect URL exemption list
SECURE_REDIRECT_EXEMPT = (
    r'healthz/$',          # Allowing HTTP access to health check
)
# Session cookie age (in seconds)
SESSION_COOKIE_AGE = 1209600

# Some security headers
SECURE_BROWSER_XSS_FILTER = True
X_FRAME_OPTIONS = 'DENY'
SECURE_CONTENT_TYPE_NOSNIFF = True

# Optionally enable HSTS
SECURE_HSTS_SECONDS = 0
SECURE_HSTS_PRELOAD = False
SECURE_HSTS_INCLUDE_SUBDOMAINS = False

# URL of login
LOGIN_URL = '{0}/accounts/login/'.format(URL_PREFIX)

# URL of logout
LOGOUT_URL = '{0}/accounts/logout/'.format(URL_PREFIX)

# Default location for login
LOGIN_REDIRECT_URL = '{0}/'.format(URL_PREFIX)

# Anonymous user name
ANONYMOUS_USER_NAME = 'anonymous'

# Reverse proxy settings
IP_PROXY_HEADER = 'HTTP_X_FORWARDED_FOR'
IP_BEHIND_REVERSE_PROXY = False
IP_PROXY_OFFSET = 0

# Sending HTML in mails
EMAIL_SEND_HTML = True

# Subject of emails includes site title
EMAIL_SUBJECT_PREFIX = '[{0}] '.format(SITE_TITLE)

# Enable remote hooks
ENABLE_HOOKS = True

# Number of nearby messages to show in each direction
NEARBY_MESSAGES = 5

# By default the length of a given translation is limited to the length of
# the source string * 10 characters. Set this option to False to allow longer
# translations (up to 10.000 characters)
LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH = True

# Use simple language codes for default language/country combinations
SIMPLIFY_LANGUAGES = True
```

(continues on next page)

(continued from previous page)

```

# Render forms using bootstrap
CRISPY_TEMPLATE_PACK = 'bootstrap3'

# List of quality checks
# CHECK_LIST = (
#     'weblate.checks.same.SameCheck',
#     'weblate.checks.chars.BeginNewlineCheck',
#     'weblate.checks.chars.EndNewlineCheck',
#     'weblate.checks.chars.BeginSpaceCheck',
#     'weblate.checks.chars.EndSpaceCheck',
#     'weblate.checks.chars.EndStopCheck',
#     'weblate.checks.chars.EndColonCheck',
#     'weblate.checks.chars.EndQuestionCheck',
#     'weblate.checks.chars.EndExclamationCheck',
#     'weblate.checks.chars.EndEllipsisCheck',
#     'weblate.checks.chars.EndSemicolonCheck',
#     'weblate.checks.chars.MaxLengthCheck',
#     'weblate.checks.chars.KashidaCheck',
#     'weblate.checks.format.PythonFormatCheck',
#     'weblate.checks.format.PythonBraceFormatCheck',
#     'weblate.checks.format.PHPFormatCheck',
#     'weblate.checks.format.CFormatCheck',
#     'weblate.checks.format.PperlFormatCheck',
#     'weblate.checks.format.JavaScriptFormatCheck',
#     'weblate.checks.format.CSharpFormatCheck',
#     'weblate.checks.format.JavaFormatCheck',
#     'weblate.checks.format.JavaMessageFormatCheck',
#     'weblate.checks.angularjs.AngularJSInterpolationCheck',
#     'weblate.checks.consistency.PluralsCheck',
#     'weblate.checks.consistency.SamePluralsCheck',
#     'weblate.checks.consistency.ConsistencyCheck',
#     'weblate.checks.consistency.TranslatedCheck',
#     'weblate.checks.chars.NewlineCountingCheck',
#     'weblate.checks.markup.BBCodeCheck',
#     'weblate.checks.chars.ZeroWidthSpaceCheck',
#     'weblate.checks.render.MaxSizeCheck',
#     'weblate.checks.markup.XMLValidityCheck',
#     'weblate.checks.markup.XMLTagsCheck',
#     'weblate.checks.markup.MarkdownRefLinkCheck',
#     'weblate.checks.markup.MarkdownLinkCheck',
#     'weblate.checks.markup.MarkdownSyntaxCheck',
#     'weblate.checks.markup.URLCheck',
#     'weblate.checks.source.OptionalPluralCheck',
#     'weblate.checks.source.EllipsisCheck',
#     'weblate.checks.source.MultipleFailingCheck',
# )

# List of automatic fixups
# AUTOFIX_LIST = (
#     'weblate.trans.autofixes.whitespace.SameBookendingWhitespace',
#     'weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis',
#     'weblate.trans.autofixes.chars.RemoveZeroSpace',
#     'weblate.trans.autofixes.chars.RemoveControlChars',
# )

# List of enabled addons
# WEBLATE_ADDONS = (

```

(continues on next page)

(continued from previous page)

```

# 'weblate.addons.gettext.GenerateMoAddon',
# 'weblate.addons.gettext.UpdateLinguasAddon',
# 'weblate.addons.gettext.UpdateConfigureAddon',
# 'weblate.addons.gettext.MsgmergeAddon',
# 'weblate.addons.gettext.GettextCustomizeAddon',
# 'weblate.addons.gettext.GettextAuthorComments',
# 'weblate.addons.cleanup.CleanupAddon',
# 'weblate.addons.consistency.LanguaugeConsistencyAddon',
# 'weblate.addons.discovery.DiscoveryAddon',
# 'weblate.addons.flags.SourceEditAddon',
# 'weblate.addons.flags.TargetEditAddon',
# 'weblate.addons.flags.SameEditAddon',
# 'weblate.addons.generate.GenerateFileAddon',
# 'weblate.addons.json.JSONCustomizeAddon',
# 'weblate.addons.properties.PropertiesSortAddon',
# 'weblate.addons.git.GitSquashAddon',
# 'weblate.addons.removal.RemoveComments',
# 'weblate.addons.removal.RemoveSuggestions',
# )

# E-mail address that error messages come from.
SERVER_EMAIL = 'noreply@example.com'

# Default email address to use for various automated correspondence from
# the site managers. Used for registration emails.
DEFAULT_FROM_EMAIL = 'noreply@example.com'

# List of URLs your site is supposed to serve
ALLOWED_HOSTS = []

# Example configuration for caching
# CACHES = {
#     'default': {
#         'BACKEND': 'django_redis.cache.RedisCache',
#         'LOCATION': 'redis://127.0.0.1:6379/0',
#         # If redis is running on same host as Weblate, you might
#         # want to use unix sockets instead:
#         # 'LOCATION': 'unix:///var/run/redis/redis.sock?db=0',
#         'OPTIONS': {
#             'CLIENT_CLASS': 'django_redis.client.DefaultClient',
#             'PARSER_CLASS': 'redis.connection.HiredisParser',
#         }
#     },
#     'avatar': {
#         'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
#         'LOCATION': os.path.join(DATA_DIR, 'avatar-cache'),
#         'TIMEOUT': 3600,
#         'OPTIONS': {
#             'MAX_ENTRIES': 1000,
#         },
#     }
# }

# REST framework settings for API
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.

```

(continues on next page)

(continued from previous page)

```

'DEFAULT_PERMISSION_CLASSES': [
    'rest_framework.permissions.IsAuthenticatedOrReadOnly'
],
'DEFAULT_AUTHENTICATION_CLASSES': (
    'rest_framework.authentication.TokenAuthentication',
    'weblate.api.authentication.BearerAuthentication',
    'rest_framework.authentication.SessionAuthentication',
),
'DEFAULT_THROTTLE_CLASSES': (
    'rest_framework.throttling.AnonRateThrottle',
    'rest_framework.throttling.UserRateThrottle'
),
'DEFAULT_THROTTLE_RATES': {
    'anon': '100/day',
    'user': '1000/day'
},
'DEFAULT_PAGINATION_CLASS': (
    'rest_framework.pagination.PageNumberPagination'
),
'PAGE_SIZE': 20,
'VIEW_DESCRIPTION_FUNCTION': 'weblate.api.views.get_view_description',
'UNAUTHENTICATED_USER': 'weblate.auth.models.get_anonymous',
}

# Example for restricting access to logged in users
# LOGIN_REQUIRED_URLS = (
#     r'/(.*)$',
# )

# In such case you will want to include some of the exceptions
# LOGIN_REQUIRED_URLS_EXCEPTIONS = (
#     r'/accounts/(.*)$',          # Required for login
#     r'/admin/login/(.*)$',       # Required for admin login
#     r'/static/(.*)$',           # Required for development mode
#     r'/widgets/(.*)$',          # Allowing public access to widgets
#     r'/data/(.*)$',             # Allowing public access to data exports
#     r'/hooks/(.*)$',            # Allowing public access to notification hooks
#     r'/healthz/$',              # Allowing public access to health check
#     r'/api/(.*)$',              # Allowing access to API
#     r'/js/i18n/$',              # JavaScript localization
#     r'/contact/$',              # Optional for contact form
#     r'/legal/(.*)$',            # Optional for legal app
# )

# Silence some of the Django system checks
SILENCED_SYSTEM_CHECKS = [
    # We have modified django.contrib.auth.middleware.AuthenticationMiddleware
    # as weblate.accounts.middleware.AuthenticationMiddleware
    'admin.E408',
]

# Celery worker configuration for testing
CELERY_TASK_ALWAYS_EAGER = True
CELERY_BROKER_URL = 'memory://'
CELERY_TASK_EAGER_PROPAGATES = True
# Celery worker configuration for production
# CELERY_TASK_ALWAYS_EAGER = False

```

(continues on next page)

(continued from previous page)

```
# CELERY_BROKER_URL = 'redis://localhost:6379'
# CELERY_RESULT_BACKEND = CELERY_BROKER_URL

# Celery settings, it is not recommended to change these
CELERY_WORKER_PREFETCH_MULTIPLIER = 0
CELERY_WORKER_MAX_MEMORY_PER_CHILD = 200000
CELERY_BEAT_SCHEDULE_FILENAME = os.path.join(
    DATA_DIR, 'celery', 'beat-schedule'
)
CELERY_TASK_ROUTES = {
    'weblate.trans.search.*': {'queue': 'search'},
    'weblate.trans.tasks.optimize_fulltext': {'queue': 'search'},
    'weblate.trans.tasks.cleanup_fulltext': {'queue': 'search'},
    'weblate.memory.tasks.*': {'queue': 'memory'},
    'weblate.accounts.tasks.notify_change': {'queue': 'notify'},
}
```

4.32 Management commands

Note: Running management commands under a different user than is running your webserver can cause wrong permissions on some files, please check *Filesystem permissions* for more details.

Django comes with a management script (available as `./manage.py` in sources or installed as **weblate** when Weblate is installed). It provides various management commands and Weblate extends it with several additional commands.

4.32.1 Invoking management commands

As mentioned before, invocation depends on how you have installed Weblate.

If you are using source code directly (either tarball or Git checkout), the management script is `./manage.py` in Weblate sources. Execution can be done as:

```
python ./manage.py list_versions
```

If you've installed Weblate using PIP installer or by `./setup.py` script, the **weblate** is installed to your path and you can use it to control Weblate:

```
weblate list_versions
```

For Docker image, the script is installed same as above, you can execute it using **docker exec**:

```
docker exec --user weblate <container> weblate list_versions
```

With **docker-compose** this is quite similar, you just have to use **docker-compose exec**:

```
docker-compose exec --user weblate weblate weblate list_versions
```

In case you need to pass some file, you can temporary add a volume:

```
docker-compose exec --user weblate /tmp:/tmp weblate weblate importusers /tmp/users.
↪ json
```

See also:

Installing using Docker, Installing Weblate with pip

4.32.2 add_suggestions

manage.py add_suggestions <project> <component> <language> <file>

New in version 2.5.

Imports translation from the file as a suggestion to given translation. It skips translations which are the same as existing ones, only different ones are added.

--author USER@EXAMPLE.COM

Email of author for the suggestions. This user has to exist prior importing (you can create one in the admin interface if needed).

Example:

```
./manage.py --author michal@cihar.com add_suggestions weblate master cs /tmp/
↪ suggestions-cs.po
```

4.32.3 auto_translate

manage.py auto_translate <project> <component> <language>

New in version 2.5.

Performs automatic translation based on other component translations.

--source PROJECT/COMPONENT

Specifies component to use as source for translation. If not specified all components in the project are used.

--user USERNAME

Specify username who will be author of the translations. Anonymous user is used if not specified.

--overwrite

Whether to overwrite existing translations.

--inconsistent

Whether to overwrite existing translations which are inconsistent (see *Inconsistent*).

--add

Automatically add language if given translation does not exist.

--mt MT

Use machine translation instead of other components.

--threshold THRESHOLD

Similarity threshold for machine translation, defaults to 80.

Example:

```
./manage.py --user nijel --inconsistent --source phpmyadmin/master phpmyadmin 4-5 cs
```

See also:

Automatic translation

4.32.4 celery_queues

manage.py celery_queues

New in version 3.7.

Displays length of Celery task queues.

4.32.5 changesite

manage.py changesite

New in version 2.4.

You can use this to change or display site name from command line without using admin interface.

--set-name NAME

Sets name for the site.

--get-name

Prints currently configured site name.

See also:

Set correct sitename

4.32.6 checkgit

manage.py checkgit <project|project/component>

Prints current state of the backend git repository.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

4.32.7 commitgit

manage.py commitgit <project|project/component>

Commits any possible pending changes to backend git repository.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

4.32.8 commit_pending

manage.py commit_pending <project|project/component>

Commits pending changes older than given age.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

--age HOURS

Age in hours for committing. If not specified value configured in *Component configuration* is used.

Note: This is automatically performed in the background by Weblate, so there is not much reason to invoke this manually besides forcing earlier commit than specified by *Component configuration*.

See also:

Running maintenance tasks, *COMMIT_PENDING_HOURS*

4.32.9 cleanup_avatar_cache

New in version 3.1.

manage.py cleanup_avatar_cache

Removes invalid items in avatar cache. This can be useful when switching between Python 2 and 3 as the cache files might be not compatible.

4.32.10 cleanuptrans

manage.py cleanuptrans

Cleanups orphaned checks and translation suggestions. This is normally not needed to execute manually, the cleanups happen automatically in the background.

See also:

Running maintenance tasks

4.32.11 createadmin

manage.py createadmin

Creates admin account with random password unless it is specified.

--password PASSWORD

Provide password on the command line and skip generating random one.

--no-password

Do not set password, this can be useful with `--update`.

--username USERNAME

Use given name instead of admin.

--email USER@EXAMPLE.COM

Specify admin email.

--name

Specify admin name (visible).

--update

Update existing user (you can use this to change password).

Changed in version 2.9: Added parameters `--username`, `--email`, `--name` and `--update`.

4.32.12 delete_memory

manage.py delete_memory

New in version 2.20.

Deletes entries in the Weblate Translation Memory.

--origin ORIGIN

Origin to delete, for imported files the origin is filename without path.

--all

Delete complete memory content and recreate the database.

See also:

Translation Memory

4.32.13 dump_memory

manage.py dump_memory

New in version 2.20.

Export a JSON file with the Weblate Translation Memory content.

See also:

Translation Memory

4.32.14 dumpuserdata

manage.py dumpuserdata <file.json>

Dumps userdata to file for later use by *importuserdata*

This is useful when migrating or merging Weblate instances.

4.32.15 import_json

manage.py import_json <json-file>

New in version 2.7.

Batch import of components based on JSON data.

The imported JSON file structure pretty much corresponds to the component object (see *GET /api/components/(string:project)/(string:component)/*). You always have to include fields name and filemask.

--project PROJECT

Specifies where the components will be imported.

--main-component COMPONENT

Use VCS repository from this component for all.

--ignore

Skip already imported components.

--update

Update already imported components.

Changed in version 2.9: Added parameters `--ignore` and `--update` to deal with already imported components.

Example of JSON file:

```
[
  {
    "slug": "po",
    "name": "Gettext PO",
    "file_format": "po",
    "filemask": "po/*.po",
    "new_lang": "none"
  },
  {
    "name": "Android",
    "filemask": "android/values-*/strings.xml",
    "template": "android/values/strings.xml",
    "repo": "weblate://test/test",
    "file_format": "aresource"
  }
]
```

See also:

[*import_memory*](#)

4.32.16 import_memory

manage.py import_memory <file>

New in version 2.20.

Imports a TMX or JSON file into the Weblate Translation Memory.

--language-map LANGMAP

Allows to map languages in the TMX to Weblate one. The language codes are mapped after normalization usually done by Weblate.

For example `--language-map en_US:en` will import all `en_US` strings as `en` ones.

This can be useful in case your TMX file locales does not match what you use in Weblate.

See also:

[*Translation Memory*](#)

4.32.17 import_project

manage.py import_project <project> <gitrepo> <branch> <filemask>

Changed in version 3.0: The `import_project` command is now based on the [*Component discovery*](#) addon and that has lead to some changes in behavior and accepted parameters.

Batch imports components into project based on file mask.

`<project>` names an existing project, into which the components should be imported.

The `<gitrepo>` defines URL of Git repository to use, and `<branch>` the git branch. To import additional translation components, from an existing Weblate component, use a `weblate://<project>/<component>` URL for the `<gitrepo>`.

The `<filemask>` defines files discovery in the repository. It can be either simple using wildcards or it can use full power of regular expressions.

The simple matching uses `**` for component name and `*` for language, for example: `**/*.po`

The regular expression has to contain named groups *component* and *language*. For example: `(?P<language>[^\s]*)/(?P<component>[^\s]*)\.po`

The import matches existing components based on files and adds the ones which do not exist. It does no changes to the already existing ones.

--name-template TEMPLATE

Customize the component's name, using Django template syntax.

For example: Documentation: `{{ component }}`

--base-file-template TEMPLATE

Customize base file for monolingual translations.

For example: `{{ component }}/res/values/string.xml`

--new-base-template TEMPLATE

Customize base file for adding new translations.

For example: `{{ component }}/ts/en.ts`

--file-format FORMAT

You can also specify file format to use (see [Supported file formats](#)), the default is autodetection.

--language-regex REGEX

You can specify language filtering (see [Component configuration](#)) by this parameter. It has to be valid regular expression.

--main-component

You can specify which component will be chosen as main - the one actually containing VCS repository.

--license NAME

Specify translation license.

--license-url URL

Specify translation license URL.

--vcs NAME

In case you need to specify version control system to use, you can do it here. The default version control is Git.

To give you some examples, let's try importing two projects.

As first we import The Debian Handbook translations, where each language has separate folder with translations of each chapter:

```
./manage.py import_project \
    debian-handbook \
    git://anonscm.debian.org/debian-handbook/debian-handbook.git \
    squeeze/master \
    '*/**.po'
```

Another example can be Tanaguru tool, where we need to specify file format, base file template and has all components and translations located in single folder:

```
./manage.py import_project \
    --file-format=properties \
    --base-file-template=web-app/tgol-web-app/src/main/resources/i18n/%s-I18N.
    properties \
    tanaguru \
    https://github.com/Tanaguru/Tanaguru \
```

(continues on next page)

(continued from previous page)

```
master \
web-app/tgol-web-app/src/main/resources/il8n/**-I18N_*.properties
```

Example of more complex parsing of filenames to get correct component and language out of filename like `src/security/Numerous_security_holes_in_0.10.1.de.po`:

```
./manage.py import_project \
tails \
git://git.tails.boum.org/tails master \
'wiki/src/security/(?P<component>.*)\.(?P<language>[^\.]*)\.po$'
```

Filtering only translations in chosen language:

```
./manage import_project \
--language-regex '^ (cs|sk)$' \
weblate \
https://github.com/WeblateOrg/weblate.git \
'weblate/locale/*/LC_MESSAGES/**/*.po'
```

See also:

More detailed examples can be found in the *Starting with internationalization* chapter, alternatively you might want to use *import_json*.

4.32.18 importuserdata

manage.py importuserdata <file.json>

Imports userdata from file created by *dumpuserdata*

4.32.19 importusers

manage.py importusers --check <file.json>

Imports users from JSON dump of Django `auth_users` database.

--check

With this option it will just check whether given file can be imported and report possible conflicts on usernames or emails.

You can dump users from existing Django installation using:

```
./manage.py dumpdata auth.User > users.json
```

4.32.20 install_addon

New in version 3.2.

manage.py install_addon --addon ADDON <project|project/component>

Installs addon to set of components.

--addon ADDON

Name of addon to install. For example `weblate.gettext.customize`.

--configuration CONFIG

JSON encoded configuration of an addon.

--update

Update existing addon configuration.

You can either define on which project or component to install addon (eg. `weblate/master`) or use `--all` to include all existing components.

For example installing *Customize gettext output* to all components:

```
./manage.py install_addon --addon weblate.gettext.customize --config '{"width": -1}' -  
↪-update --all
```

See also:

Addons

4.32.21 list_ignored_checks

manage.py list_ignored_checks

Lists most frequently ignored checks. This can be useful for tuning your setup, if users have to ignore too many of consistency checks.

4.32.22 list_languages

manage.py list_languages <locale>

Lists supported language in MediaWiki markup - language codes, English names and localized names.

This is used to generate `<https://wiki.l10n.cz/Jazyky>`.

4.32.23 list_memory

manage.py list_memory

New in version 2.20.

Lists contents of the Weblate Translation Memory.

--type {origin}

Type of information to list, defaults to listing used origins.

See also:

Translation Memory

4.32.24 list_translators

manage.py list_translators <project|project/component>

Renders the list of translators by language for the given project:

```
[French]  
Jean Dupont <jean.dupont@example.com>  
[English]  
John Doe <jd@example.com>
```

--language-code

Use language code instead of language name in output.

You can either define which project or component to use (eg. `weblate/master`) or use `--all` to list translators from all existing components.

4.32.25 list_versions

manage.py list_versions

Lists versions of Weblate dependencies.

4.32.26 loadpo

manage.py loadpo <project|project/component>

Reloads translations from disk (eg. in case you did some updates in VCS repository).

--force

Force update even if the files should be up to date.

--lang LANGUAGE

Limit processing to single language.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

Note: You seldom need to invoke this, Weblate will automatically load changed files on VCS update. This is needed in case you manually change underlying Weblate VCS repository or in some special cases after upgrade.

4.32.27 lock_translation

manage.py lock_translation <project|project/component>

Locks given component for translating. This is useful in case you want to do some maintenance on underlying repository.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

See also:

`unlock_translation`

4.32.28 move_language

manage.py move_language source target

New in version 3.0.

Allows you to merge language content. This is useful when updating to new version which contains aliases for previously unknown languages which were created with the (*generated*) suffix. It moves all content from the *source* language to *target* one.

Example:

```
./manage.py move_language cze cs
```

After moving the content, you should review if there is nothing left (this is subject to race conditions when somebody updates the repository meanwhile) and remove the (*generated*) language.

4.32.29 optimize_memory

manage.py optimize_memory

New in version 3.2.

Optimizes translation memory storage.

--rebuild

The index will be completely rebuilt by dumping all content and creating it again. It is recommended to backup it prior to this operation.

See also:

Translation Memory, Backing up and moving Weblate, dump_memory

4.32.30 pushgit

manage.py pushgit <project|project/component>

Pushes committed changes to upstream VCS repository.

--force-commit

Force committing any pending changes prior to push.

You can either define which project or component to update (eg. *weblate/master*) or use *--all* to update all existing components.

Note: Weblate does push changes automatically if *Push on commit* in *Component configuration* is enabled, what is default.

4.32.31 rebuild_index

manage.py rebuild_index <project|project/component>

Rebuilds index for fulltext search. This might be lengthy operation if you have a huge set of translation strings.

--clean

Removes all words from database prior updating, this is implicit when called with *--all*.

--optimize

The index will not be processed again, only its content will be optimized (removing stale entries and merging possibly split index files).

See also:

Fulltext search

4.32.32 unlock_translation

manage.py unlock_translation <project|project/component>

Unlocks a given component for translating. This is useful in case you want to do some maintenance on the underlying repository.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

See also:

lock_translation

4.32.33 setupgroups

manage.py setupgroups

Configures default groups and optionally assigns all users to default group.

--no-privs-update

Disables update of existing groups (only adds new ones).

--no-projects-update

Prevents updates of groups for existing projects. This allows to add newly added groups to existing projects, see *Per project access control*.

See also:

Access control

4.32.34 setuplang

manage.py setuplang

Setups list of languages (it has own list and all defined in `translate-toolkit`).

--no-update

Disables update of existing languages (only adds new ones).

4.32.35 updatechecks

manage.py updatechecks <project|project/component>

Updates all check for all strings. This could be useful only on upgrades which do major changes to checks.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

4.32.36 updategit

manage.py updategit <project|project/component>

Fetches remote VCS repositories and updates internal cache.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

Note: Usually it is better to configure hooks in the repository to trigger *Notification hooks* instead of regular polling by *updategit*.

4.33 Whiteboard messages

You can use whiteboard messages to give some information to your translators. The message can be site-wide or targeted to a translation component or language.

This can be useful for various things from announcing the purpose of the website to specifying targets for translations.

The whiteboard can currently be specified only in the admin interface:

The screenshot shows the 'Add Whiteboard message' form in the Weblate administration interface. The header bar is dark blue with 'Weblate administration' on the left and navigation links on the right. Below the header, a breadcrumb trail shows 'Home > Weblate translations > Whiteboard messages > Add Whiteboard message'. The form itself is titled 'Add Whiteboard message' and includes a note that required fields are marked in bold. A large text area for the message is shown with the example text 'Translations will be used only if they reach 60%'. Below this is a checkbox for 'Render as HTML' with a note that URLs will be converted to links and markup will be escaped. The form has several dropdown menus for 'Project' (set to 'WeblateOrg'), 'Component', 'Language', and 'Category' (set to 'Info (light blue)'). There is also an 'Expiry date' field set to 'Today' with a calendar icon. At the bottom, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

The whiteboard messages are then shown based on specified context:

No context specified

Shown on dashboard (landing page).

Project specified

Shown on project, all its components and translations.

Component specified

Shown on component and all its translations.

Language specified

Shown on language overview and all translations in this language.

You can see how it looks on the language overview page:

Project	Strings	Strings of total	Words	Words of total	Needs editing	Checks	Suggestions	Comments
WeblateOrg	97.9%	97.9%	94.2%	94.2%	0.0%	6.2%	0.0%	0.0%

And on the project page:

Component	Strings	Words	Needs editing	Checks	Suggestions	Comments
Django	81.7%	66.5%	5.8%	4.8%	0.0%	0.0%
Language names	95.5%	95.0%	3.4%	0.0%	0.0%	0.0%

4.34 Component Lists

Weblate allows you to specify multiple lists of components. These will then appear as options on the user dashboard, and users can pick a list to be their default view when they log in. See [Dashboard](#) to learn more about this feature.

Changed in version 2.20: The overview of all component lists status is also available on the dashboard.

The names and contents of component lists can be specified in the admin interface, in *Component lists* section. Each component list must have a name that is displayed to the user, and a slug that represents it in the URL.

Note: Since version 2.13 you can also change the dashboard settings for the anonymous user in the admin interface, this will change what dashboard is visible to unauthenticated users.

4.34.1 Automatic component lists

New in version 2.13.

Additionally you can create *Automatic component list assignment* rules to automatically add components to the list based on their slug. This can be useful for maintaining component lists for large installations or in case you want to have component list with all components on your Weblate installation.

To create component list containing all components, you can simply define *Automatic component list assignment* with `^.*$` regular expression on both project and component as shown on following image:

Weblate administration WELCOME, WEBLATE TEST VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT

Home · Weblate translations · Component lists · Add Component list

Add Component list

Required fields are marked in bold.

Component list name: Display name

URL slug: Name used in URLs and filenames.

☒ Show on dashboard
When enabled this component list will be shown as a tab on the dashboard

Components:

Available components ⓘ

Filter

- WeblateOrg/Django
- WeblateOrg/Language names

Choose all ⓘ

Chosen components ⓘ

Remove all ⓘ

Hold down "Control", or "Command" on a Mac, to select more than one.

AUTOMATIC COMPONENT LIST ASSIGNMENTS		
PROJECT REGULAR EXPRESSION ⓘ	COMPONENT REGULAR EXPRESSION ⓘ	DELETE?
<input type="text" value="^.*\$"/>	<input type="text" value="^.*\$"/>	<input type="button" value="✕"/>

[+ Add another Automatic component list assignment](#)

4.35 Optional Weblate modules

Weblate comes with several optional modules which might be useful for your setup.

4.35.1 Git exporter

New in version 2.10.

The Git exporter provides you read only access to the underlying Git repository using HTTP.

Installation

To install, simply add `weblate.gitexport` to installed applications in `settings.py`:

```
INSTALLED_APPS += (
    'weblate.gitexport',
)
```

After installing, you need to migrate your database so that existing repositories are properly exported:

```
./manage.py migrate
```

Usage

The module automatically hooks into Weblate and sets exported repository URL in the *Component configuration*. The repositories are accessible under `/git/` path of the Weblate, for example `https://example.org/git/weblate/master/`:

```
git clone 'https://example.org/git/weblate/master/'
```

Repositories are available anonymously unless *Per project access control* is enabled. In that case you need to authenticate using your API token (you can obtain it in your *User profile*):

```
git clone 'https://user:KEY@example.org/git/weblate/master/'
```

4.35.2 Billing

New in version 2.4.

Billing module is used on *Hosted Weblate* and is used to define billing plans, track invoices and usage limits.

Installation

To install, simply add `weblate.billing` to installed applications in `settings.py`:

```
INSTALLED_APPS += (
    'weblate.billing',
)
```

This module includes additional database structures, to have them installed you should run the database migration:

```
./manage.py migrate
```

Usage

After installation you can control billing in the admin interface. Users with billing enabled will get new *Billing* tab in their *User profile*.

The billing module additionally allows project admins to create new projects and components without being superusers (see *Adding translation projects and components*). This is possible when following conditions are met:

- The billing is in it's configured limits (any overusage results in blocking of project/component creation) and paid (if it's price is non zero)

- The user is admin of existing project with billing or user is owner of billing (the latter is necessary when creating new billing for users to be able to import new projects).

Upon project creation user is able to choose which billing should be charged for the project in case he has access to more of them.

4.35.3 Legal

New in version 2.15.

Legal module is used on [Hosted Weblate](#) and is used to provide required legal documents. It comes with blank documents and you are expected to provide following templates with the documents:

legal/documents/tos.html Terms of service document

legal/documents/privacy.html Privacy policy document

legal/documents/summary.html Short overview of terms of service and privacy policy

Note: You can find legal documents for the Hosted Weblate service in separate Git repository <<https://github.com/WeblateOrg/hosted/tree/master/wlhosted/templates/legal/documents>>.

Most likely these will not be directly usable for you, but you might want to use them as a starting point and adjust them to match your use case.

Installation

To install, simply add `weblate.legal` to installed applications in `settings.py`:

```
INSTALLED_APPS += (
    'weblate.legal',
)

# Optionals:

# Social auth pipeline to confirm TOS on registration/login
SOCIAL_AUTH_PIPELINE += (
    'weblate.legal.pipeline.tos_confirm',
)

# Middleware to enforce TOS confirmation of logged in users
MIDDLEWARE += [
    'weblate.legal.middleware.RequireTOSMiddleware',
]
```

This module includes additional database structures, to have them installed you should run the database migration:

```
./manage.py migrate
```

Now you should edit the legal documents to match your service. You can find them in the `weblate/legal/templates/legal/` folder.

Usage

After installation the legal documents are shown in Weblate UI.

4.35.4 Avatars

Weblate comes with built in support for showing user avatars based on emails. This can be disabled using `ENABLE_AVATARS`. The avatars are downloaded and cached server side to reduce information leaks to the sites serving them.

Weblate currently supports single backend:

- Gravatar

See also:

Avatar caching, `AVATAR_URL_PREFIX`, `ENABLE_AVATARS`

4.35.5 Spam protection

Optionally Weblate can be protected against suggestion spamming by unauthenticated users through akismet.com service.

To enable this, you need to install *akismet* Python module and configure Akismet API key.

See also:

`AKISMET_API_KEY`


4.35.6 Signing Git commits by GnuPG

New in version 3.1.

Weblate allows you to sign all commits by it's GnuPG key. To configure this, you need to enable `WEBLATE_GPG_IDENTITY`. Weblate will generate GnuPG key when needed and will use it to sign all translation commits.

This feature needs GnuPG 2.1 or newer installed.

You can find the key in the `DATA_DIR` and the public key is shown on the about page:


[Dashboard](#)
[Projects](#)
[Languages](#)

[Register](#)
[Login](#)

About Weblate

About

Statistics


Keys

About Weblate

Weblate is a web based translation tool with tight version control integration. It features a simple and clean user interface, propagation of translations across components, quality checks and automatic linking to source files.

More information about Weblate can be found on weblate.org.

Weblate is libre software created by volunteers, and accepts donations.


[Donate to Weblate](#)

Versions

Weblate	3.7.1
Python	3.7.3
Django	2.2
Celery	4.3.0
celery-batches	0.2
six	1.12.0
social-auth-core	3.1.0
social-auth-app-django	3.1.0
django-appconf	1.0.3
translate-toolkit	2.3.1
translation-finder	1.5
Whoosh	2.7.4
defusedxml	0.6.0
Git	2.20.1
Pillow	6.0.0
python-dateutil	2.8.0
lxml	4.3.3
django-crispy-forms	1.7.2
django_compressor	2.2
djanoorestframework	3.9.2
user-agents	2.0
jellyfish	0.7.1
diff-match-patch	20121119
pytz	2019.1
pyuca	1.2
ruamel.yaml	0.15.94
tesseractocr	2.4.0
Mercurial	4.9
git-svn	2.20.1
git-review	1.28.0

Weblate owes its existence to these projects.

Powered by Weblate 3.7.1
 [About Weblate](#)
[Legal](#)
[Contact](#)
[Documentation](#)
[Donate to Weblate](#)

Alternatively you can also import existing keys into Weblate, just set `HOME=$DATA_DIR/home` when invoking `gpg`.

See also:

WEBLATE_GPG_IDENTITY

4.35.7 Rate limiting

Changed in version 3.2: The rate limiting now accepts more fine grained configuration.

Several operations in Weblate are rate limited. At most `RATELIMIT_ATTEMPTS` attempts are allowed within `RATELIMIT_WINDOW` seconds. The user is then blocked for `RATELIMIT_LOCKOUT`. There are also per scope variants of those settings, eg. `RATELIMIT_CONTACT_ATTEMPTS` or `RATELIMIT_TRANSLATE_ATTEMPTS`, see table below for full list of available scopes.

Following operations are subject to rate limiting:

Name	Scope	Allowed tempts	at- dow	Ratelimit win- dow	Lockout pe- riod
Registration	REGISTRATION	5		300	600
Sending message to admins	MESSAGE	5		300	600
Password authentication on login	LOGIN	5		300	600
Sitewide search	SEARCH	6		60	60
Translating	TRANSLATE	30		60	600
Adding to glossary	GLOSSARY	30		60	600

Additionally if there are more than `AUTH_LOCK_ATTEMPTS` failed authentication attempts on one account, this account password authentication is disabled and it's not possible to login until user asks for password reset.

See also:

Rate limiting

IP address for rate limiting

The rate limiting is based on client IP address. This is obtained from HTTP headers and you will have to change configuration in the event Weblate is running behind reverse proxy to work it properly.

See also:

`IP_BEHIND_REVERSE_PROXY`, `IP_PROXY_HEADER`, `IP_PROXY_OFFSET`

4.36 Customizing Weblate

Weblate can be extended or customized using standard Django and Python ways. Always please consider contributing changes upstream so that everybody can benefit from your additions. Including your changes in Weblate itself will also reduce your maintenance costs - code in Weblate is taken care of when changing internal interfaces or refactoring the code.

Warning: Neither internal interfaces or templates are considered as stable API. Please review your customizations on every upgrade, the interface or their semantics might change without notice.

See also:

Contributing

4.36.1 Creating Python module

If you are not familiar with Python, you might want to look into [Python For Beginners](#) which explains the basics and will point you to further tutorials.

We're about to write custom Python code and we need place to store it - it's called a module in Python. Later we need to place somewhere where the Python interpreter can import it - either in system path (usually something like `/usr/lib/python3.7/site-packages/`) or in Weblate directory, which is also added to the interpreter search path.

The best approach is to create a proper Python package out of your customization:

1. Place your Python module with check into folder which will match your package name. We're using *weblate_customization* in following examples.
2. Add empty `__init__.py` file to the same directory. This ensures Python can import this whole package.
3. Write `setup.py` in parent directory to describe your package:

```
from setuptools import setup

setup(
    name = "weblate_customization",
    version = "0.0.1",
    author = "Michal Cihar",
    author_email = "michal@cihar.com",
    description = "Sample Custom check for Weblate.",
    license = "BSD",
    keywords = "weblate check example",
    packages=['weblate_customization'],
)
```

4. Now you can install it using `pip install -e .`
5. Once installed into Python path, you can use it in Weblate configuration, in most cases as fully qualified path (for example `weblate_customization.checks.FooCheck`).

Overall your module structure should look like:

```
weblate_customization
├── setup.py
└── weblate_customization
    ├── __init__.py
    ├── addons.py
    └── checks.py
```

You can find example application for customizing Weblate at <https://github.com/WeblateOrg/customize-example>, it covers all topics described below.

4.36.2 Changing logo

To change logo you need to create simple Django app which will contain static files which you want to overwrite (see [Creating Python module](#)). Then you add it into `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # Weblate apps are here...

    # Add your customization as last
```

(continues on next page)

(continued from previous page)

```
'weblate_customization',
)
```

And then execute `./manage.py collectstatic --noinput`, this will collect static files served to clients.

See also:

Managing static files (e.g. images, JavaScript, CSS), *Serving static files*

4.36.3 Custom quality checks and auto fixes

You have implemented code for *Custom automatic fixups* or *Customizing behavior* and now it's time to install it into Weblate. First place them into your Python module with Weblate customization (see *Creating Python module*). Then enabled it is just matter of adding its fully-qualified path to Python class to appropriate settings (`CHECK_LIST` or `AUTOFIX_LIST`):

```
CHECK_LIST = (
    'weblate_customization.checks.FooCheck',
)
```

See also:

Writing own checks

4.36.4 Custom addons

First place them into your Python module with Weblate customization (see *Creating Python module*). Then enabled it is just matter of adding its fully-qualified path to Python class to appropriate settings (`WEBLATE_ADDONS`):

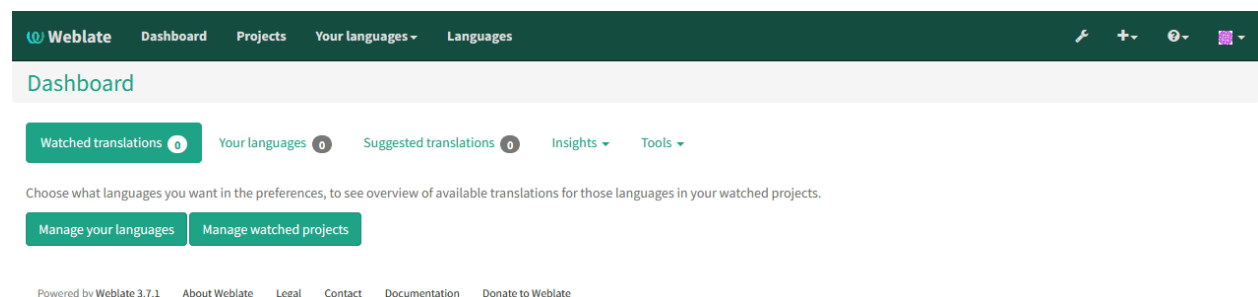
```
WEBLATE_ADDONS = (
    'weblate_customization.addons.ExamplePreAddon',
)
```

See also:

Writing addon, Executing scripts from addon

4.37 Django admin interface

Administration of Weblate is done through standard Django admin interface, which is available under `/admin/` URL. Once logged in as user with proper privileges, you can access it using the wrench icon in top navigation:



Here you can manage objects stored in the database, such as users, translations and other settings:

Webplate administration

WELCOME, WEBLATE TEST. [VIEW SITE](#) / [DOCUMENTATION](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration

REPORTS

Status of repositories

SSH keys

Performance report

Translation memory

ACCOUNTS

Audit logs

+ Add

Change

Profiles

+ Add

Change

Verified emails

+ Add

Change

AUTH TOKEN

Tokens

+ Add

Change

AUTHENTICATION

Automatic group assignments

+ Add

Change

Groups

+ Add

Change

Roles

+ Add

Change

Users

+ Add

Change

BILLING

Billings

+ Add

Change

Invoices

+ Add

Change

Plans

+ Add

Change

FONTS

Font groups

+ Add

Change

Fonts

+ Add

Change

LEGAL

Agreements

+ Add

Change

PYTHON SOCIAL AUTH

Associations

+ Add

Change

Nonces

+ Add

Change

User social auths

+ Add

Change

SCREENSHOTS

Screenshots

+ Add

Change

SITES

Sites

+ Add

Change

WEBLATE LANGUAGES

Languages

+ Add

Change

WEBLATE TRANSLATIONS

Component lists

+ Add

Change

Components

+ Add

Change

Contributor agreements

+ Add

Change

Projects

+ Add

Change

Whiteboard messages

+ Add

Change

Recent actions

My actions

None available

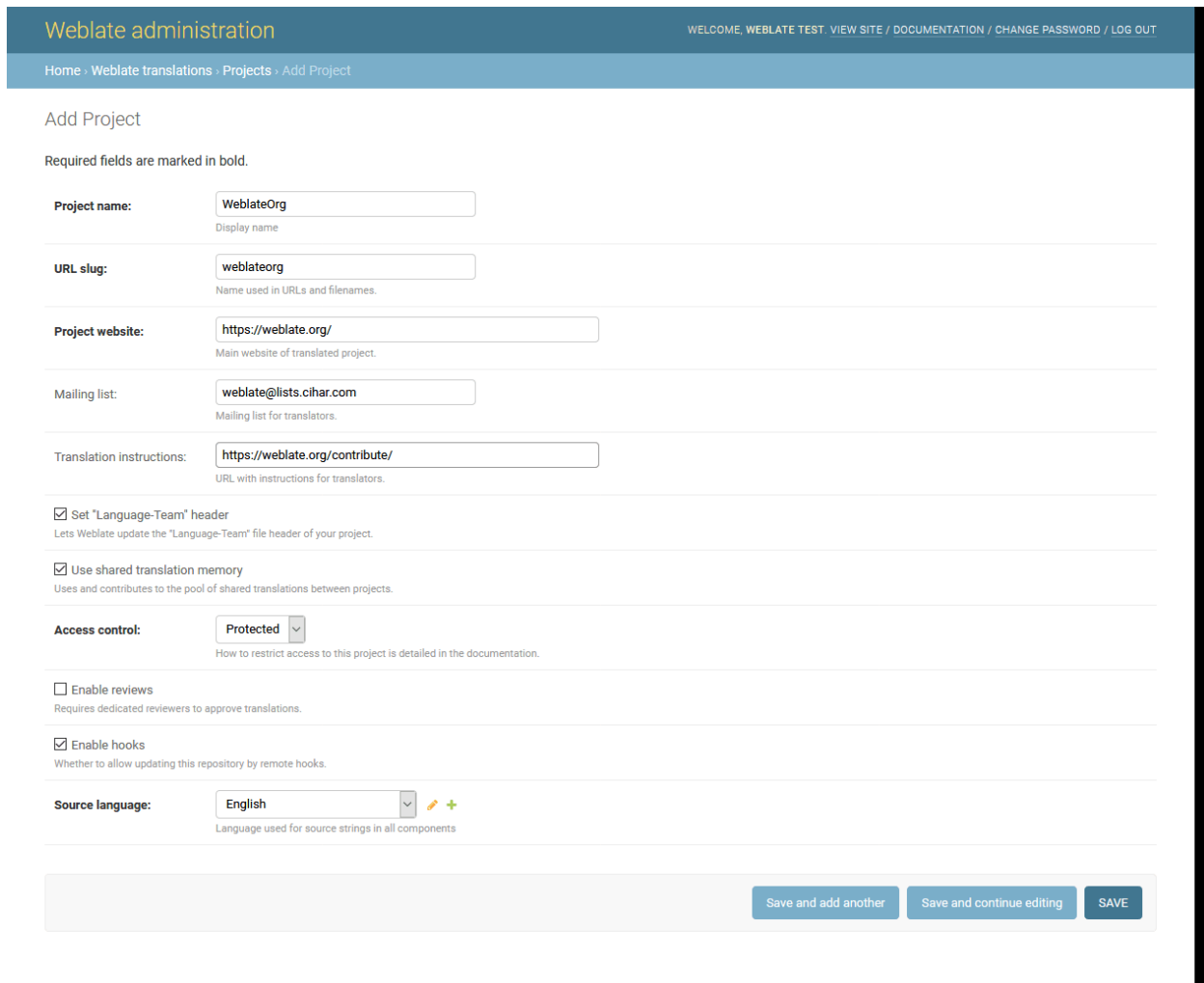
In the *Reports* section you can check the status of your site, tweak it for *Production setup* or manage SSH keys to access *Accessing repositories*.

With all sections below you can manage database objects. The most interesting one is probably *Weblate translations*, where you can manage translatable projects, see *Project configuration* and *Component configuration*.

Another section, *Weblate languages* holds language definitions, see *Language definitions* for more details.

4.37.1 Adding project

First you have to add project, which will serve as container for all components. Usually you create one project for one piece of software or book (see *Project configuration* for information on individual parameters):



The screenshot shows the 'Weblate administration' interface. At the top, there's a navigation bar with 'Home', 'Weblate translations', 'Projects', and 'Add Project'. The main heading is 'Add Project'. Below it, a note says 'Required fields are marked in bold.' The form contains several fields: 'Project name' (WeblateOrg), 'URL slug' (weblateorg), 'Project website' (https://weblate.org/), 'Mailing list' (weblate@lists.cihar.com), and 'Translation instructions' (https://weblate.org/contribute/). There are also checkboxes for 'Set "Language-Team" header', 'Use shared translation memory', 'Enable reviews', and 'Enable hooks'. An 'Access control' dropdown is set to 'Protected'. At the bottom, there's a 'Source language' dropdown set to 'English'. Three buttons are at the bottom right: 'Save and add another', 'Save and continue editing', and 'SAVE'.

See also:

Project configuration

4.37.2 Bilingual components

Once you have added a project, you can add translation components to it (see *Component configuration* for information on individual parameters):

Weblate administration

WELCOME, WEBLATE TESTVIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT

Home › Weblate translations › Components › Add Component

Add Component

IMPORT SPEED DOCUMENTATION

Required fields are marked in bold.

Component name:

Language names

Display name

URL slug:

language-names

Name used in URLs and filenames.

Project:

WeblateOrg

Version control system:

Git

Version control system to use to access your repository containing translations. You can also choose additional integration with third party providers to submit merge requests.

Source code repository:

https://github.com/WeblateOrg/demo.git

URL of a repository, use weblate://project/component to share it with other component.

Repository push URL:

URL of a push repository, pushing is turned off if empty.

Repository browser:

github.com/WeblateOrg/demo/blob/({branch})/({filename})#L({line})

Link to repository browser, use ({branch}) for branch, ({filename}) and ({line}) as filename and line placeholders.

Exported repository URL:

URL of repository where users can fetch changes from Weblate

Source string bug reporting address:

Email address for reports on errors in source strings. Leave empty for no emails.

Repository branch:

Repository branch to translate

Filemask:

/langdata/locale/*/LC_MESSAGES/django.po

Path of files to translate relative to repository root, use * instead of language code, for example: po/*-po or locale/*/LC_MESSAGES/django.po.

Monolingual base language file:

Filename of translation base file, containing all strings and their source; it is recommended for monolingual translation formats.

☒ Edit base file

Whether users will be able to edit the base file for monolingual translations.

Base file for new translations:

weblate/langdata/locale/django.pot

Filename of file used for creating new translations. For gettext choose .pot file.

File format:

Gettext PO file

☐ Locked

Locked component will not get any translation updates.

☒ Allow translation propagation

Whether translation updates in other components will cause automatic translation in this one

☒ Save translation history

Whether Weblate should keep track of old translations.

☒ Turn on suggestions

Whether to allow translation suggestions at all.

☐ Suggestion voting

Whether users can vote for suggestions.

Autoaccept suggestions:

0

Automatically accept suggestions with this number of votes, use 0 to disable.

Translation flags:

Additional comma-separated flags to influence quality checks, check documentation for possible values.

Translation license:

GPL-3.0+

Optional short summary of license used for translations

License URL:

https://spdx.org/licenses/GPL-3.0+

Optional URL with license details.

Contributor agreement:

See also:

Component configuration, Bilingual and monolingual formats

4.37.3 Monolingual components

For easier translating of monolingual formats, you should provide a template file, which contains mapping of message IDs to source language (usually English) (see [Component configuration](#) for information on individual parameters):

Weblate administration

WELCOME, WEBLATE TEST. [VIEW SITE](#) / [DOCUMENTATION](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

[Home](#) · [Weblate translations](#) · [Components](#) · [Add Component](#)

IMPORT SPEED DOCUMENTATION

Add Component

Required fields are marked in bold.

Component name:

Android

Display name

URL slug:

android

Name used in URLs and filenames.

Project:

WeblateOrg

Version control system:

Git

Version control system to use to access your repository containing translations. You can also choose additional integration with third party providers to submit merge requests.

Source code repository:

weblate://weblateorg/language-names

URL of a repository, use weblate://project/component to share it with other component.

Repository push URL:

URL of a push repository, pushing is turned off if empty.

Repository browser:

Link to repository browser, use {{branch}} for branch, {{filename}} and {{line}} as filename and line placeholders.

Exported repository URL:

URL of repository where users can fetch changes from Weblate

Source string bug reporting address:

Email address for reports on errors in source strings. Leave empty for no emails.

Repository branch:

Repository branch to translate

Filemask:

app/src/main/res/values-*/strings.xml

Path of files to translate relative to repository root, use * instead of language code, for example: po/*-po or locale/*/LC_MESSAGES/django.po.

Monolingual base language file:

app/src/main/res/values/strings.xml

Filename of translation base file, containing all strings and their source; it is recommended for monolingual translation formats.

☒ Edit base file

Whether users will be able to edit the base file for monolingual translations.

Base file for new translations:

Filename of file used for creating new translations. For gettext choose .pot file.

File format:

Android String Resource

☐ Locked

Locked component will not get any translation updates.

☒ Allow translation propagation

Whether translation updates in other components will cause automatic translation in this one

☒ Save translation history

Whether Weblate should keep track of old translations.

☒ Turn on suggestions

Whether to allow translation suggestions at all.

☐ Suggestion voting

Whether users can vote for suggestions.

Autoaccept suggestions:

0

Automatically accept suggestions with this number of votes, use 0 to disable.

Translation flags:

Additional comma-separated flags to influence quality checks, check documentation for possible values.

Translation license:

MIT

Optional short summary of license used for translations

License URL:

https://spdx.org/licenses/MIT

Optional URL with license details.

Contributor agreement:

See also:

Component configuration, Bilingual and monolingual formats

4.38 Changes

4.38.1 weblate 3.7.1

Released on June 28th 2019.

- Documentation updates.
- Fixed some requirements constraints.
- Updated language database.
- Localization updates.
- Various user interface tweaks.
- Improved handling of unsupported but discovered translation files.
- More verbosely report missing file format requirements.

4.38.2 weblate 3.7

Released on June 21st 2019.

- Added separate Celery queue for notifications.
- Use consistent look with application for API browsing.
- Include approved stats in the reports.
- Report progress when updating translation component.
- Allow to abort running background component update.
- Extend template language for filename manipulations.
- Use templates for editor link and repository browser URL.
- Indicate max length and current chars count when editing translation.
- Improved handling of abbreviations in unchanged translation check.
- Refreshed landing page for new contributors.
- Add support for configuring msgmerge addon.
- Delay opening SMTP connection when sending notifications.
- Improved error logging.
- Allow custom location in MO generating addon.
- Added addons to cleanup old suggestions or comments.
- Added option to enable horizontal mode in the zen editor.
- Improved import performance with many linked components.
- Fixed examples installation in some cases.
- Improved rendering of alerts in changes.

- Added new horizontal stats widget.
- Improved format strings check on plurals.
- Added font management tool.
- New check for rendered text dimensions.
- Added support for subtitle formats.
- Include overall completion stats for languages.
- Added reporting at project and global scope.
- Improved user interface when showing translation status.
- New Weblate logo and color scheme.
- New look of bitmap badges.

4.38.3 weblate 3.6.1

Released on April 26th 2019.

- Improved handling of monolingual Xliff files.
- Fixed digest notifications in some corner cases.
- Fixed addon script error alert.
- Fixed generating MO file for monolingual PO files.
- Fixed display of uninstalled checks.
- Indicate administered projects on project listing.
- Allow update to recover from missing VCS repository.

4.38.4 weblate 3.6

Released on April 20th 2019.

- Add support for downloading user data.
- Addons are now automatically triggered upon installation.
- Improved instructions for resolving merge conflicts.
- Cleanup addon is now compatible with app store metadata translations.
- Configurable language code syntax when adding new translations.
- Warn about using Python 2 with planned termination of support in April 2020.
- Extract special chars from the source string for visual keyboard.
- Extended contributor stats to reflect both source and target counts.
- Admins and consistency addons can now add translations even if disabled for users.
- Fixed description of toggle disabling Language–Team header manipulation.
- Notify users mentioned in comments.
- Removed file format autodetection from component setup.
- Fixed generating MO file for monolingual PO files.

- Added digest notifications.
- Added support for muting component notifications.
- Added notifications for new alerts, whiteboard messages or components.
- Notifications for administered projects can now be configured.
- Improved handling of three letter language codes.

4.38.5 weblate 3.5.1

Released on March 10th 2019.

- Fixed Celery systemd unit example.
- Fixed notifications from http repositories with login.
- Fixed race condition in editing source string for monolingual translations.
- Include output of failed addon execution in the logs.
- Improved validation of choices for adding new language.
- Allow to edit file format in component settings.
- Update installation instructions to prefer Python 3.
- Performance and consistency improvements for loading translations.
- Make Microsoft Terminology service compatible with current zeep releases.
- Localization updates.

4.38.6 weblate 3.5

Released on March 3rd 2019.

- Improved performance of built in translation memory.
- Added interface to manage global translation memory.
- Improved alerting on bad component state.
- Added user interface to manage whiteboard messages.
- Addon commit message now can be configured.
- Reduce number of commits when updating upstream repository.
- Fixed possible metadata loss when moving component between projects.
- Improved navigation in the zen mode.
- Added several new quality checks (Markdown related and URL).
- Added support for app store metadata files.
- Added support for toggling GitHub or Gerrit integration.
- Added check for Kashida letters.
- Added option to squash commits based on authors.
- Improved support for xlsx file format.
- Compatibility with tesseract 4.0.

- Billing addon now removes projects for unpaid billings after 45 days.

4.38.7 weblate 3.4

Released on January 22nd 2019.

- Added support for XLIFF placeholders.
- Celery can now utilize multiple task queues.
- Added support for renaming and moving projects and components.
- Include chars counts in reports.
- Added guided adding of translation components with automatic detection of translation files.
- Customizable merge commit messages for Git.
- Added visual indication of component alerts in navigation.
- Improved performance of loading translation files.
- New addon to squash commits prior to push.
- Improved displaying of translation changes.
- Changed default merge style to rebase and made that configurable.
- Better handle private use subtags in language code.
- Improved performance of fulltext index updates.
- Extended file upload API to support more parameters.

4.38.8 weblate 3.3

Released on November 30th 2018.

- Added support for component and project removal.
- Improved performance for some monolingual translations.
- Added translation component alerts to highlight problems with a translation.
- Expose XLIFF string rename as context when available.
- Added support for XLIFF states.
- Added check for non writable files in DATA_DIR.
- Improved CSV export for changes.

4.38.9 weblate 3.2.2

Released on October 20th 2018.

- Remove no longer needed Babel dependency.
- Updated language definitions.
- Improve documentation for addons, LDAP and Celery.
- Fixed enabling new dos-eol and auto-java-messageformat flags.

- Fixed running setup.py test from PyPI package.
- Improved plurals handling.
- Fixed translation upload API failure in some corner cases.
- Fixed updating Git configuration in case it was changed manually.

4.38.10 weblate 3.2.1

Released on October 10th 2018.

- Document dependency on backports.csv on Python 2.7.
- Fix running tests under root.
- Improved error handling in gitexport module.
- Fixed progress reporting for newly added languages.
- Correctly report Celery worker errors to Sentry.
- Fixed creating new translations with Qt Linguist.
- Fixed occasional fulltext index update failures.
- Improved validation when creating new components.
- Added support for cleanup of old suggestions.

4.38.11 weblate 3.2

Released on October 6th 2018.

- Add install_addon management command for automated addon installation.
- Allow more fine grained ratelimit settings.
- Added support for export and import of Excel files.
- Improve component cleanup in case of multiple component discovery addons.
- Rewritten Microsoft Terminology machine translation backend.
- Weblate now uses Celery to offload some processing.
- Improved search capabilities and added regular expression search.
- Added support for Youdao Zhiyun API machine translation.
- Added support for Baidu API machine translation.
- Integrated maintenance and cleanup tasks using Celery.
- Improved performance of loading translations by almost 25%.
- Removed support for merging headers on upload.
- Removed support for custom commit messages.
- Configurable editing mode (zen/full).
- Added support for error reporting to Sentry.
- Added support for automated daily update of repositories.
- Added support for creating projects and components by users.

- Built in translation memory now automatically stores translations done.
- Users and projects can import their existing translation memories.
- Better management of related strings for screenshots.
- Added support for checking Java MessageFormat.

See [3.2 milestone on GitHub](#) for detailed list of addressed issues.

4.38.12 weblate 3.1.1

Released on July 27th 2018.

- Fix testsuite failure on some setup.

4.38.13 weblate 3.1

Released on July 27th 2018.

- Upgrades from older version than 3.0.1 are not supported.
- Allow to override default commit messages from settings.
- Improve webhooks compatibility with self hosted environments.
- Added support for Amazon Translate.
- Compatibility with Django 2.1.
- Django system checks are now used to diagnose problems with installation.
- Removed support for soon shutdown libavatar service.
- New addon to mark unchanged translations as needing edit.
- Add support for jumping to specific location while translating.
- Downloaded translations can now be customized.
- Improved calculation of string similarity in translation memory matches.
- Added support by signing Git commits by GnuPG.

4.38.14 weblate 3.0.1

Released on June 10th 2018.

- Fixed possible migration issue from 2.20.
- Localization updates.
- Removed obsolete hook examples.
- Improved caching documentation.
- Fixed displaying of admin documentation.
- Improved handling of long language names.

4.38.15 weblate 3.0

Released on June 1st 2018.

- Rewritten access control.
- Several code cleanups that lead to moved and renamed modules.
- New addon for automatic component discovery.
- The `import_project` management command has now slightly different parameters.
- Added basic support for Windows RC files.
- New addon to store contributor names in PO file headers.
- The per component hook scripts are removed, use addons instead.
- Add support for collecting contributor agreements.
- Access control changes are now tracked in history.
- New addon to ensure all components in a project have same translations.
- Support for more variables in commit message templates.
- Add support for providing additional textual context.

4.38.16 weblate 2.20

Released on April 4th 2018.

- Improved speed of cloning subversion repositories.
- Changed repository locking to use third party library.
- Added support for downloading only strings needing action.
- Added support for searching in several languages at once.
- New addon to configure Gettext output wrapping.
- New addon to configure JSON formatting.
- Added support for authentication in API using RFC 6750 compatible Bearer authentication.
- Added support for automatic translation using machine translation services.
- Added support for HTML markup in whiteboard messages.
- Added support for mass changing state of strings.
- Translate-toolkit at least 2.3.0 is now required, older versions are no longer supported.
- Added built in translation memory.
- Added componentlists overview to dashboard and per component list overview pages.
- Added support for DeepL machine translation service.
- Machine translation results are now cached inside Weblate.
- Added support for reordering committed changes.

4.38.17 weblate 2.19.1

Released on February 20th 2018.

- Fixed migration issue on upgrade from 2.18.
- Improved file upload API validation.

4.38.18 weblate 2.19

Released on February 15th 2018.

- Fixed imports across some file formats.
- Display human friendly browser information in audit log.
- Added TMX exporter for files.
- Various performance improvements for loading translation files.
- Added option to disable access management in Weblate in favor of Django one.
- Improved glossary lookup speed for large strings.
- Compatibility with django_auth_ldap 1.3.0.
- Configuration errors are now stored and reported persistently.
- Honor ignore flags in whitespace autofixer.
- Improved compatibility with some Subversion setups.
- Improved built in machine translation service.
- Added support for SAP Translation Hub service.
- Added support for Microsoft Terminology service.
- Removed support for advertisement in notification mails.
- Improved translation progress reporting at language level.
- Improved support for different plural formulas.
- Added support for Subversion repositories not using stdlayout.
- Added addons to customize translation workflows.

4.38.19 weblate 2.18

Released on December 15th 2017.

- Extended contributor stats.
- Improved configuration of special chars virtual keyboard.
- Added support for DTD file format.
- Changed keyboard shortcuts to less likely collide with browser/system ones.
- Improved support for approved flag in XLIFF files.
- Added support for not wrapping long strings in Gettext po files.
- Added button to copy permalink for current translation.

- Dropped support for Django 1.10 and added support for Django 2.0.
- Removed locking of translations while translating.
- Added support for adding new strings to monolingual translations.
- Added support for translation workflows with dedicated reviewers.

4.38.20 weblate 2.17.1

Released on October 13th 2017.

- Fixed running testsuite in some specific situations.
- Locales updates.

4.38.21 weblate 2.17

Released on October 13th 2017.

- Weblate by default does shallow Git clones now.
- Improved performance when updating large translation files.
- Added support for blocking certain emails from registration.
- Users can now delete their own comments.
- Added preview step to search and replace feature.
- Client side persistence of settings in search and upload forms.
- Extended search capabilities.
- More fine grained per project ACL configuration.
- Default value of BASE_DIR has been changed.
- Added two step account removal to prevent accidental removal.
- Project access control settings is now editable.
- Added optional spam protection for suggestions using Akismet.

4.38.22 weblate 2.16

Released on August 11th 2017.

- Various performance improvements.
- Added support for nested JSON format.
- Added support for WebExtension JSON format.
- Fixed git exporter authentication.
- Improved CSV import in certain situations.
- Improved look of Other translations widget.
- The max-length checks is now enforcing length of text in form.
- Make the commit_pending age configurable per component.

- Various user interface cleanups.
- Fixed component/project/sitewide search for translations.

4.38.23 weblate 2.15

Released on June 30th 2017.

- Show more related translations in other translations.
- Add option to see translations of current string to other languages.
- Use 4 plural forms for Lithuanian by default.
- Fixed upload for monolingual files of different format.
- Improved error messages on failed authentication.
- Keep page state when removing word from glossary.
- Added direct link to edit secondary language translation.
- Added Perl format quality check.
- Added support for rejecting reused passwords.
- Extended toolbar for editing RTL languages.

4.38.24 weblate 2.14.1

Released on May 24th 2017.

- Fixed possible error when paginating search results.
- Fixed migrations from older versions in some corner cases.
- Fixed possible CSRF on project watch and unwatch.
- The password reset no longer authenticates user.
- Fixed possible captcha bypass on forgotten password.

4.38.25 weblate 2.14

Released on May 17th 2017.

- Add glossary entries using AJAX.
- The logout now uses POST to avoid CSRF.
- The API key token reset now uses POST to avoid CSRF.
- Weblate sets Content-Security-Policy by default.
- The local editor URL is validated to avoid self-XSS.
- The password is now validated against common flaws by default.
- Notify users about important activity with their account such as password change.
- The CSV exports now escape potential formulas.
- Various minor improvements in security.

- The authentication attempts are now rate limited.
- Suggestion content is stored in the history.
- Store important account activity in audit log.
- Ask for password confirmation when removing account or adding new associations.
- Show time when suggestion has been made.
- There is new quality check for trailing semicolon.
- Ensure that search links can be shared.
- Included source string information and screenshots in the API.
- Allow to overwrite translations through API upload.

4.38.26 weblate 2.13.1

Released on Apr 12th 2017.

- Fixed listing of managed projects in profile.
- Fixed migration issue where some permissions were missing.
- Fixed listing of current file format in translation download.
- Return HTTP 404 when trying to access project where user lacks privileges.

4.38.27 weblate 2.13

Released on Apr 12th 2017.

- Fixed quality checks on translation templates.
- Added quality check to trigger on losing translation.
- Add option to view pending suggestions from user.
- Add option to automatically build component lists.
- Default dashboard for unauthenticated users can be configured.
- Add option to browse 25 random strings for review.
- History now indicates string change.
- Better error reporting when adding new translation.
- Added per language search within project.
- Group ACLs can now be limited to certain permissions.
- The per project ACLs are now implemented using Group ACL.
- Added more fine grained privileges control.
- Various minor UI improvements.

4.38.28 weblate 2.12

Released on Mar 3rd 2017.

- Improved admin interface for groups.
- Added support for Yandex Translate API.
- Improved speed of sitewide search.
- Added project and component wide search.
- Added project and component wide search and replace.
- Improved rendering of inconsistent translations.
- Added support for opening source files in local editor.
- Added support for configuring visual keyboard with special characters.
- Improved screenshot management with OCR support for matching source strings.
- Default commit message now includes translation information and URL.
- Added support for Joomla translation format.
- Improved reliability of import across file formats.

4.38.29 weblate 2.11

Released on Jan 31st 2017.

- Include language detailed information on language page.
- Mercurial backend improvements.
- Added option to specify translation component priority.
- More consistent usage of Group ACL even with less used permissions.
- Added WL_BRANCH variable to hook scripts.
- Improved developer documentation.
- Better compatibility with various Git versions in Git exporter addon.
- Included per project and component stats.
- Added language code mapping for better support of Microsoft Translate API.
- Moved fulltext cleanup to background job to make translation removal faster.
- Fixed displaying of plural source for languages with single plural form.
- Improved error handling in import_project.
- Various performance improvements.

4.38.30 weblate 2.10.1

Released on Jan 20th 2017.

- Do not leak account existence on password reset form (CVE-2017-5537).

4.38.31 weblate 2.10

Released on Dec 15th 2016.

- Added quality check to check whether plurals are translated differently.
- Fixed GitHub hooks for repositories with authentication.
- Added optional Git exporter module.
- Support for Microsoft Cognitive Services Translator API.
- Simplified project and component user interface.
- Added automatic fix to remove control chars.
- Added per language overview to project.
- Added support for CSV export.
- Added CSV download for stats.
- Added matrix view for quick overview of all translations
- Added basic API for changes and strings.
- Added support for Apertium APy server for machine translations.

4.38.32 weblate 2.9

Released on Nov 4th 2016.

- Extended parameters for createadmin management command.
- Extended import_json to be able to handle with existing components.
- Added support for YAML files.
- Project owners can now configure translation component and project details.
- Use “Watched” instead of “Subscribed” projects.
- Projects can be watched directly from project page.
- Added multi language status widget.
- Highlight secondary language if not showing source.
- Record suggestion deletion in history.
- Improved UX of languages selection in profile.
- Fixed showing whiteboard messages for component.
- Keep preferences tab selected after saving.
- Show source string comment more prominently.
- Automatically install Gettext PO merge driver for Git repositories.
- Added search and replace feature.
- Added support for uploading visual context (screenshots) for translations.

4.38.33 weblate 2.8

Released on Aug 31st 2016.

- Documentation improvements.
- Translations.
- Updated bundled javascript libraries.
- Added list_translators management command.
- Django 1.8 is no longer supported.
- Fixed compatibility with Django 1.10.
- Added Subversion support.
- Separated XML validity check from XML mismatched tags.
- Fixed API to honor HIDE_REPO_CREDENTIALS settings.
- Show source change in zen mode.
- Alt+PageUp/PageDown/Home/End now works in zen mode as well.
- Add tooltip showing exact time of changes.
- Add option to select filters and search from translation page.
- Added UI for translation removal.
- Improved behavior when inserting placeables.
- Fixed auto locking issues in zen mode.

4.38.34 weblate 2.7

Released on Jul 10th 2016.

- Removed Google web translate machine translation.
- Improved commit message when adding translation.
- Fixed Google Translate API for Hebrew language.
- Compatibility with Mercurial 3.8.
- Added import_json management command.
- Correct ordering of listed translations.
- Show full suggestion text, not only a diff.
- Extend API (detailed repository status, statistics, ...).
- Testsuite no longer requires network access to test repositories.

4.38.35 weblate 2.6

Released on Apr 28th 2016.

- Fixed validation of components with language filter.
- Improved support for XLIFF files.

- Fixed machine translation for non English sources.
- Added REST API.
- Django 1.10 compatibility.
- Added categories to whiteboard messages.

4.38.36 weblate 2.5

Released on Mar 10th 2016.

- Fixed automatic translation for project owners.
- Improved performance of commit and push operations.
- New management command to add suggestions from command line.
- Added support for merging comments on file upload.
- Added support for some GNU extensions to C printf format.
- Documentation improvements.
- Added support for generating translator credits.
- Added support for generating contributor stats.
- Site wide search can search only in one language.
- Improve quality checks for Armenian.
- Support for starting translation components without existing translations.
- Support for adding new translations in Qt TS.
- Improved support for translating PHP files.
- Performance improvements for quality checks.
- Fixed sitewide search for failing checks.
- Added option to specify source language.
- Improved support for XLIFF files.
- Extended list of options for import_project.
- Improved targeting for whiteboard messages.
- Support for automatic translation across projects.
- Optimized fulltext search index.
- Added management command for auto translation.
- Added placeables highlighting.
- Added keyboard shortcuts for placeables, checks and machine translations.
- Improved translation locking.
- Added quality check for AngularJS interpolation.
- Added extensive group based ACLs.
- Clarified terminology on strings needing review (formerly fuzzy).
- Clarified terminology on strings needing action and not translated strings.

- Support for Python 3.
- Dropped support for Django 1.7.
- Dropped dependency on msginit for creating new Gettext po files.
- Added configurable dashboard views.
- Improved notifications on parse errors.
- Added option to import components with duplicate name to import_project.
- Improved support for translating PHP files
- Added XLIFF export for dictionary.
- Added XLIFF and Gettext PO export for all translations.
- Documentation improvements.
- Added support for configurable automatic group assignments.
- Improved adding of new translations.

4.38.37 weblate 2.4

Released on Sep 20th 2015.

- Improved support for PHP files.
- Ability to add ACL to anonymous user.
- Improved configurability of import_project command.
- Added CSV dump of history.
- Avoid copy/paste errors with whitespace chars.
- Added support for Bitbucket webhooks.
- Tigher control on fuzzy strings on translation upload.
- Several URLs have changed, you might have to update your bookmarks.
- Hook scripts are executed with VCS root as current directory.
- Hook scripts are executed with environment variables describing current component.
- Add management command to optimize fulltext index.
- Added support for error reporting to Rollbar.
- Projects now can have multiple owners.
- Project owners can manage themselves.
- Added support for javascript-format used in Gettext PO.
- Support for adding new translations in XLIFF.
- Improved file format autodetection.
- Extended keyboard shortcuts.
- Improved dictionary matching for several languages.
- Improved layout of most of pages.
- Support for adding words to dictionary while translating.

- Added support for filtering languages to be managed by Weblate.
- Added support for translating and importing CSV files.
- Rewritten handling of static files.
- Direct login/registration links to third party service if that's the only one.
- Commit pending changes on account removal.
- Add management command to change site name.
- Add option to configure default committer.
- Add hook after adding new translation.
- Add option to specify multiple files to add to commit.

4.38.38 weblate 2.3

Released on May 22nd 2015.

- Dropped support for Django 1.6 and South migrations.
- Support for adding new translations when using Java Property files
- Allow to accept suggestion without editing.
- Improved support for Google OAuth2.
- Added support for Microsoft .resx files.
- Tuned default robots.txt to disallow big crawling of translations.
- Simplified workflow for accepting suggestions.
- Added project owners who always receive important notifications.
- Allow to disable editing of monolingual template.
- More detailed repository status view.
- Direct link for editing template when changing translation.
- Allow to add more permissions to project owners.
- Allow to show secondary language in zen mode.
- Support for hiding source string in favor of secondary language.

4.38.39 weblate 2.2

Released on Feb 19th 2015.

- Performance improvements.
- Fulltext search on location and comments fields.
- New SVG/javascript based activity charts.
- Support for Django 1.8.
- Support for deleting comments.
- Added own SVG badge.
- Added support for Google Analytics.

- Improved handling of translation filenames.
- Added support for monolingual JSON translations.
- Record component locking in a history.
- Support for editing source (template) language for monolingual translations.
- Added basic support for Gerrit.

4.38.40 weblate 2.1

Released on Dec 5th 2014.

- Added support for Mercurial repositories.
- Replaced Glyphicon font by Awesome.
- Added icons for social authentication services.
- Better consistency of button colors and icons.
- Documentation improvements.
- Various bugfixes.
- Automatic hiding of columns in translation listing for small screens.
- Changed configuration of filesystem paths.
- Improved SSH keys handling and storage.
- Improved repository locking.
- Customizable quality checks per source string.
- Allow to hide completed translations from dashboard.

4.38.41 weblate 2.0

Released on Nov 6th 2014.

- New responsive UI using Bootstrap.
- Rewritten VCS backend.
- Documentation improvements.
- Added whiteboard for site wide messages.
- Configurable strings priority.
- Added support for JSON file format.
- Fixed generating mo files in certain cases.
- Added support for GitLab notifications.
- Added support for disabling translation suggestions.
- Django 1.7 support.
- ACL projects now have user management.
- Extended search possibilities.
- Give more hints to translators about plurals.

- Fixed Git repository locking.
- Compatibility with older Git versions.
- Improved ACL support.
- Added buttons for per language quotes and other special chars.
- Support for exporting stats as JSONP.

4.38.42 weblate 1.9

Released on May 6th 2014.

- Django 1.6 compatibility.
- No longer maintained compatibility with Django 1.4.
- Management commands for locking/unlocking translations.
- Improved support for Qt TS files.
- Users can now delete their account.
- Avatars can be disabled.
- Merged first and last name attributes.
- Avatars are now fetched and cached server side.
- Added support for shields.io badge.

4.38.43 weblate 1.8

Released on November 7th 2013.

- Please check manual for upgrade instructions.
- Nicer listing of project summary.
- Better visible options for sharing.
- More control over anonymous users privileges.
- Supports login using third party services, check manual for more details.
- Users can login by email instead of username.
- Documentation improvements.
- Improved source strings review.
- Searching across all strings.
- Better tracking of source strings.
- Captcha protection for registration.

4.38.44 weblate 1.7

Released on October 7th 2013.

- Please check manual for upgrade instructions.
- Support for checking Python brace format string.
- Per component customization of quality checks.
- Detailed per translation stats.
- Changed way of linking suggestions, checks and comments to strings.
- Users can now add text to commit message.
- Support for subscribing on new language requests.
- Support for adding new translations.
- Widgets and charts are now rendered using Pillow instead of Pango + Cairo.
- Add status badge widget.
- Dropped invalid text direction check.
- Changes in dictionary are now logged in history.
- Performance improvements for translating view.

4.38.45 weblate 1.6

Released on July 25th 2013.

- Nicer error handling on registration.
- Browsing of changes.
- Fixed sorting of machine translation suggestions.
- Improved support for MyMemory machine translation.
- Added support for Amagama machine translation.
- Various optimizations on frequently used pages.
- Highlights searched phrase in search results.
- Support for automatic fixups while saving the message.
- Tracking of translation history and option to revert it.
- Added support for Google Translate API.
- Added support for managing SSH host keys.
- Various form validation improvements.
- Various quality checks improvements.
- Performance improvements for import.
- Added support for voting on suggestions.
- Cleanup of admin interface.

4.38.46 weblate 1.5

Released on April 16th 2013.

- Please check manual for upgrade instructions.
- Added public user pages.
- Better naming of plural forms.
- Added support for TBX export of glossary.
- Added support for Bitbucket notifications.
- Activity charts are now available for each translation, language or user.
- Extended options of import_project admin command.
- Compatible with Django 1.5.
- Avatars are now shown using libravatar.
- Added possibility to pretty print JSON export.
- Various performance improvements.
- Indicate failing checks or fuzzy strings in progress bars for projects or languages as well.
- Added support for custom pre-commit hooks and committing additional files.
- Rewritten search for better performance and user experience.
- New interface for machine translations.
- Added support for monolingual po files.
- Extend amount of cached metadata to improve speed of various searches.
- Now shows word counts as well.

4.38.47 weblate 1.4

Released on January 23rd 2013.

- Fixed deleting of checks/comments on string deletion.
- Added option to disable automatic propagation of translations.
- Added option to subscribe for merge failures.
- Correctly import on projects which needs custom ttkit loader.
- Added sitemaps to allow easier access by crawlers.
- Provide direct links to string in notification emails or feeds.
- Various improvements to admin interface.
- Provide hints for production setup in admin interface.
- Added per language widgets and engage page.
- Improved translation locking handling.
- Show code snippets for widgets in more variants.
- Indicate failing checks or fuzzy strings in progress bars.

- More options for formatting commit message.
- Fixed error handling with machine translation services.
- Improved automatic translation locking behaviour.
- Support for showing changes from previous source string.
- Added support for substring search.
- Various quality checks improvements.
- Support for per project ACL.
- Basic string tests coverage.

4.38.48 weblate 1.3

Released on November 16th 2012.

- Compatibility with PostgreSQL database backend.
- Removes languages removed in upstream git repository.
- Improved quality checks processing.
- Added new checks (BB code, XML markup and newlines).
- Support for optional rebasing instead of merge.
- Possibility to relocate Weblate (eg. to run it under /weblate path).
- Support for manually choosing file type in case autodetection fails.
- Better support for Android resources.
- Support for generating SSH key from web interface.
- More visible data exports.
- New buttons to enter some special characters.
- Support for exporting dictionary.
- Support for locking down whole Weblate installation.
- Checks for source strings and support for source strings review.
- Support for user comments for both translations and source strings.
- Better changes log tracking.
- Changes can now be monitored using RSS.
- Improved support for RTL languages.

4.38.49 weblate 1.2

Released on August 14th 2012.

- Weblate now uses South for database migration, please check upgrade instructions if you are upgrading.
- Fixed minor issues with linked git repos.
- New introduction page for engaging people with translating using Weblate.
- Added widgets which can be used for promoting translation projects.

- Added option to reset repository to origin (for privileged users).
- Project or component can now be locked for translations.
- Possibility to disable some translations.
- Configurable options for adding new translations.
- Configuration of git commits per project.
- Simple antispam protection.
- Better layout of main page.
- Support for automatically pushing changes on every commit.
- Support for email notifications of translators.
- List only used languages in preferences.
- Improved handling of not known languages when importing project.
- Support for locking translation by translator.
- Optionally maintain Language-Team header in po file.
- Include some statistics in about page.
- Supports (and requires) django-registration 0.8.
- Caching of counted strings with failing checks.
- Checking of requirements during setup.
- Documentation improvements.

4.38.50 weblate 1.1

Released on July 4th 2012.

- Improved several translations.
- Better validation while creating component.
- Added support for shared git repositories across components.
- Do not necessary commit on every attempt to pull remote repo.
- Added support for offloading indexing.

4.38.51 weblate 1.0

Released on May 10th 2012.

- Improved validation while adding/saving component.
- Experimental support for Android component files (needs patched ttkit).
- Updates from hooks are run in background.
- Improved installation instructions.
- Improved navigation in dictionary.

4.38.52 weblate 0.9

Released on April 18th 2012.

- Fixed import of unknown languages.
- Improved listing of nearby messages.
- Improved several checks.
- Documentation updates.
- Added definition for several more languages.
- Various code cleanups.
- Documentation improvements.
- Changed file layout.
- Update helper scripts to Django 1.4.
- Improved navigation while translating.
- Better handling of po file renames.
- Better validation while creating component.
- Integrated full setup into syncdb.
- Added list of recent changes to all translation pages.
- Check for not translated strings ignores format string only messages.

4.38.53 weblate 0.8

Released on April 3rd 2012.

- Replaced own full text search with Whoosh.
- Various fixes and improvements to checks.
- New command updatechecks.
- Lot of translation updates.
- Added dictionary for storing most frequently used terms.
- Added /admin/report/ for overview of repositories status.
- Machine translation services no longer block page loading.
- Management interface now contains also useful actions to update data.
- Records log of changes made by users.
- Ability to postpone commit to Git to generate less commits from single user.
- Possibility to browse failing checks.
- Automatic translation using already translated strings.
- New about page showing used versions.
- Django 1.4 compatibility.
- Ability to push changes to remote repo from web interface.

- Added review of translations done by others.

4.38.54 weblate 0.7

Released on February 16th 2012.

- Direct support for GitHub notifications.
- Added support for cleaning up orphaned checks and translations.
- Displays nearby strings while translating.
- Displays similar strings while translating.
- Improved searching for string.

4.38.55 weblate 0.6

Released on February 14th 2012.

- Added various checks for translated messages.
- Tunable access control.
- Improved handling of translations with new lines.
- Added client side sorting of tables.
- Please check upgrading instructions in case you are upgrading.

4.38.56 weblate 0.5

Released on February 12th 2012.

- **Support for machine translation using following online services:**
 - Apertium
 - Microsoft Translator
 - MyMemory
- Several new translations.
- Improved merging of upstream changes.
- Better handle concurrent git pull and translation.
- Propagating works for fuzzy changes as well.
- Propagating works also for file upload.
- Fixed file downloads while using FastCGI (and possibly others).

4.38.57 weblate 0.4

Released on February 8th 2012.

- Added usage guide to documentation.
- Fixed API hooks not to require CSRF protection.

4.38.58 weblate 0.3

Released on February 8th 2012.

- Better display of source for plural translations.
- New documentation in Sphinx format.
- Displays secondary languages while translating.
- Improved error page to give list of existing projects.
- New per language stats.

4.38.59 weblate 0.2

Released on February 7th 2012.

- Improved validation of several forms.
- Warn users on profile upgrade.
- Remember URL for login.
- Naming of text areas while entering plural forms.
- Automatic expanding of translation area.

4.38.60 weblate 0.1

Released on February 6th 2012.

- Initial release.

4.39 About Weblate

4.39.1 Project goals

Web based localization tool with tight *Version control integration* supporting a wide range of *Supported file formats*, making it easy for translators to contribute.

4.39.2 Project name

The project is named as combination of the words “web” and “translate”.

4.39.3 Project website

You can find the project website at <<https://weblate.org/>>, there is also a demonstration server at <<https://demo.weblate.org/>>, and a hosted service at <<https://hosted.weblate.org/>>. This documentation can be browsed on <<https://docs.weblate.org/>>.

4.39.4 Leadership

This project is maintained by Michal Čihař <michal@cihar.com>.

4.39.5 Authors

Weblate was started by Michal Čihař <michal@cihar.com> and since it's start in 2012 thousands of people have contributed.

4.40 Contributing

There are dozens of ways to contribute in Weblate. Any help is welcomed, be it coding, graphics design, documentation or sponsorship.

4.40.1 Code and development

Weblate is developed on [GitHub](#). You are welcome to fork the code and open pull requests. Patches in any other form are welcome too.

See also:

Check out [Internals](#) to see how Weblate looks from inside.

4.40.2 Coding standard

The code should follow PEP-8 coding guidelines. It is recommended to format new code using **black** code formatter (though existing code is yet formatted).

To check the code quality, you can use **flake8**, the recommended plugins are listed in `requirements-test.txt`.

You can execute all coding style checks with the script `ci/run-lint`.

4.40.3 Security by Design Principles

Any code for Weblate should be writted with [Security by Design Principles](#) in mind.

4.40.4 Developer's Certificate of Origin

In contributing to the Weblate project, please certify to the following:

Weblate Developer's Certificate of Origin. Version 1.0

By making a contribution to this project, I certify that:

- (a) The contribution was created in whole or in part by me and I have the right to submit it under the license of "GNU General Public License or any later version" ("GPLv3-or-later"); or
- (b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under GPLv3-or-later; or

- (c) The contribution was provided directly to me by some other person who certified (a) or (b) and I have not modified it.
- (d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all metadata and personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with Weblate’s policies and the requirements of the GPLv2-or-later where they are relevant.
- (e) I am granting this work to this project under the terms of the GPLv3-or-later.

<https://www.gnu.org/licenses/gpl-3.0.html>

Please confirm your affirmation of the above by adding the following line to your patch:

```
Signed-off-by: Jane Developer <jane@example.org>
```

using your real name (sorry, no pseudonyms or anonymous contributions).

If you are a developer authorized to contribute to Weblate on behalf of your employer, please use your corporate email address in the “Signed-off-by” tag. If not, please use a personal email address.

4.40.5 Testsuite

Testsuites exist for most of the current code, increase coverage by adding testcases for any new functionality, and verify that it works. Current test results can be found on [Travis](#) and coverage is reported on [Codecov](#).

To run a testsuite locally, use:

```
DJANGO_SETTINGS_MODULE=weblate.settings_test ./manage.py test
```

You can also specify individual tests to run:

```
DJANGO_SETTINGS_MODULE=weblate.settings_test ./manage.py test weblate.gitexport
```

See also:

See [Testing in Django](#) for more info on running and writing tests for Django.

4.40.6 Reporting issues

Our [issue tracker](#) is hosted at GitHub:

Feel welcome to report any issues with, or suggest improvement of Weblate there. If what you have found is a security issue in Weblate, please consult the “Security issues” section below.

4.40.7 Security issues

In order to give the community time to respond and upgrade you are strongly urged to report all security issues privately. HackerOne is used to handle security issues, and can be reported directly at [HackerOne](#).

Alternatively, report to security@weblate.org, which ends up on HackerOne as well.

If you don’t want to use HackerOne, for whatever reason, you can send the report by email to michal@cihar.com. You can choose to encrypt it using this PGP key `3CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D`.

Note: Weblate depends on third party components for many things. In case you find a vulnerability affecting one of those components in general, please report it directly to the respective project.

Some of these are:

- Django
 - Django REST framework
 - Python Social Auth
-

4.40.8 Starting with our codebase

If looking for some bugs to familiarize yourself with the Weblate codebase, look for ones labelled *good first issue* <<https://github.com/WeblateOrg/weblate/labels/good%20first%20issue>>‘_:

If you have Docker and docker-compose installed, you can spin up the development environment simply by running:

```
./rundev.sh
```

4.40.9 Translating

Weblate is being [translated](#) using Weblate itself, feel free to take part in the effort of making Weblate available in as many human languages as possible.

4.40.10 Funding Weblate development

You can fund further Weblate development on the [donate page](#). Funds collected there are used to fund gratis hosting for libre software projects, and further development of Weblate. Please check the *donate page* for details, such as funding goals and rewards you can get by being a funder.

Backers who have funded Weblate

List of Weblate supporters:

- Yashiro Ccs
- Cheng-Chia Tseng
- Timon Reinhard
- [Cassidy James](#)
- Loic Dachary
- Marozed

Do you want to be in the list? Please see options on the [Donate to Weblate](#).

4.40.11 Releasing Weblate

Release checklist:

1. Make sure screenshots are up to date `make -C docs update-screenshots`
2. Create a release `./scripts/create-release --tag`
3. Push tags to GitHub

4. Update Docker image
5. Close GitHub milestone
6. Enable building version docs on Read the Docs
7. Once the Docker image is tested, add a tag and push it

4.41 Debugging Weblate

Weblate is just a software and it can contain bugs. The bugs can behave as application crashes or as a misbehavior. You are welcome to collect information on the issue and submit it to our [issue tracker](#).

4.41.1 Analyzing application crashes

In case application crashes, it is useful to collect as much information about the crash as possible. The easiest way to achieve this is by using third party services which can collect such information automatically. You can find information how to set this up in *Collecting error reports*.

4.41.2 Silent failures

Lot of tasks are offloaded to Celery for background processing. In case of failure those are not shown in the user interface, but appear in the Celery logs. Configuring *Collecting error reports* will help you to notice such failures easier.

4.41.3 Performance issues

In case Weblate preforms badly in some situation, please collect relevant logs showing the issue and which might help figuring out where our code might be improved.

In case some requests are taking too long without any indication, you might want to install *dogslow* <<https://pypi.org/project/dogslow/>> together with *Collecting error reports* and get detailed traceback of problematic places in the error collection tool.

4.42 Internals

Note: This chapter will give you basic overview of Weblate internals.

Weblate is derives most of its code structure from, and is based on Django. Familiarize yourself with Django at a [glance](#) to get a basic understanding of its file structure.

4.42.1 Modules

Weblate consists of several Django applications (some optional, see *Optional Weblate modules*):

`accounts`

User account, profiles and notifications.

addons

Addons to tweak Weblate behavior, see *Addons*.

api

API based on *Django REST* framework.

auth

Authentication and permissions.

billing

The optional *Billing* module.

formats

File format abstraction layer based on translate-toolkit.

gitexport

The optional *Git exporter* module.

lang

Module defining language and plural models.

langdata

Language data definitions.

legal

The optional *Legal* module.

machinery

Integration of machine translation services.

memory

Built in translation memory, see *Translation Memory*.

permissions

Obsolete.

screenshots

Screenshots management and OCR module.

trans

Main module handling translations.

utils

Various helper utilities.

vcs

Version control system abstraction.

wladmin

Django admin interface customization.

4.43 License

Copyright (C) 2012 - 2019 Michal Čihař <michal@cihar.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

4.44 Legal documents

Note: Herein you will find various legal information you might need to operate Weblate in certain legal jurisdictions. It is provided as a means of guidance, without any warranty of accuracy or correctness. It is ultimately your responsibility to ensure that your use of Weblate complies with all applicable laws and regulations.

4.44.1 ITAR and other export controls

Weblate can be run within your own datacenter or virtual private cloud. As such, it can be used to store ITAR or other export-controlled information, however, end users are responsible for ensuring such compliance.

The Hosted Weblate service has not been audited for compliance with ITAR or other export controls, and does not currently offer the ability to restrict translations access by country.

4.44.2 US encryption controls

Weblate does not contain any cryptographic code, but might be subject export controls as it uses third party components utilizing cryptography for authentication, data-integrity and -confidentiality.

Most likely Weblate would be classified as ECCN 5D002 or 5D992 and, as publicly available libre software, it should not be subject to EAR (see [Encryption items NOT Subject to the EAR](#)).

Software components used by Weblate (listing only components related to cryptographic function):

Python See https://wiki.python.org/moin/PythonSoftwareFoundationLicenseFaq#Is_Python_subject_to_export_laws.3F

GnuPG Optionally used by Weblate

Git Optionally used by Weblate

curl Used by Git

OpenSSL Used by Python and cURL

The strength of encryption keys depend on the configuration of Weblate and the third party components it interacts with, but in any decent setup it will include all export restricted cryptographic functions:

- In excess of 56 bits for a symmetric algorithm

- Factorisation of integers in excess of 512 bits for an asymmetric algorithm
- Computation of discrete logarithms in a multiplicative group of a finite field of size greater than 512 bits for an asymmetric algorithm
- Discrete logarithms in a group different than above in excess of 112 bits for an asymmetric algorithm

Weblate doesn't have any cryptographic activation feature, but it can be configured in a way where no cryptography code would be involved. The cryptographic features include:

- Accessing remote servers using secure protocols (HTTPS)
- Generating signatures for code commits (PGP)

See also:

[Export Controls \(EAR\) on Open Source Software](#)

W

wlc, [103](#)
wlc.config, [104](#)
wlc.main, [104](#)

HTTP Routing Table

/	GET /api/screenshots/(int:pk)/file/, 96
ANY /, 76	GET /api/sources/, 95
	GET /api/sources/(int:pk)/, 95
/api	GET /api/translations/, 87
GET /api/, 79	GET /api/translations/(string:project)/(string:component)/, 88
GET /api/changes/, 94	GET /api/translations/(string:project)/(string:component)/, 90
GET /api/changes/(int:pk)/, 94	
GET /api/components/, 83	GET /api/translations/(string:project)/(string:component)/, 90
GET /api/components/(string:project)/(string:component)/, 83	GET /api/translations/(string:project)/(string:component)/changes/, 91
GET /api/components/(string:project)/(string:component)/changes/, 85	GET /api/translations/(string:project)/(string:component)/lock/, 92
GET /api/components/(string:project)/(string:component)/lock/, 85	GET /api/translations/(string:project)/(string:component)/monolingual_base/, 90
GET /api/components/(string:project)/(string:component)/monolingual_base/, 86	GET /api/units/, 93
GET /api/components/(string:project)/(string:component)/new_template/, 87	GET /api/units/(int:pk)/, 93
GET /api/components/(string:project)/(string:component)/repository/, 86	POST /api/components/(string:project)/(string:component)/repository/, 85
GET /api/components/(string:project)/(string:component)/statistics/, 87	POST /api/components/(string:project)/(string:component)/statistics/, 86
GET /api/components/(string:project)/(string:component)/translations/, 87	POST /api/projects/(string:project)/repository/, 81
GET /api/components/(string:project)/statistics/, 83	POST /api/screenshots/(int:pk)/file/, 96
GET /api/languages/, 79	POST /api/translations/(string:project)/(string:component)/, 91
GET /api/languages/(string:language)/, 79	POST /api/translations/(string:project)/(string:component)/, 92
GET /api/projects/, 80	
GET /api/projects/(string:project)/, 80	/exports
GET /api/projects/(string:project)/changes/, 81	GET /exports/rss/, 99
GET /api/projects/(string:project)/components/, 82	GET /exports/rss/(string:project)/, 99
GET /api/projects/(string:project)/repository/, 81	GET /exports/rss/(string:project)/(string:component)/, 99
GET /api/screenshots/, 95	GET /exports/rss/(string:project)/(string:component)/, 99
GET /api/screenshots/(int:pk)/, 95	GET /exports/rss/language/(string:language)/, 99

GET /exports/stats/(string:project)/(string:component)/,
[97](#)

/hooks

GET /hooks/update/(string:project)/,[96](#)

GET /hooks/update/(string:project)/(string:component)/,
[96](#)

POST /hooks/bitbucket/,[97](#)

POST /hooks/github/,[96](#)

POST /hooks/gitlab/,[97](#)

POST /hooks/pagure/,[97](#)

Symbols

- add
 - auto_translate command line option, [247](#)
- addon ADDON
 - install_addon command line option, [253](#)
- age HOURS
 - commit_pending command line option, [248](#)
- all
 - delete_memory command line option, [250](#)
- author USER@EXAMPLE.COM
 - add_suggestions command line option, [247](#)
- base-file-template TEMPLATE
 - import_project command line option, [252](#)
- check
 - importusers command line option, [253](#)
- clean
 - rebuild_index command line option, [256](#)
- config PATH
 - wlc command line option, [100](#)
- config-section SECTION
 - wlc command line option, [100](#)
- configuration CONFIG
 - install_addon command line option, [253](#)
- convert
 - wlc command line option, [101](#)
- email USER@EXAMPLE.COM
 - createadmin command line option, [249](#)
- file-format FORMAT
 - import_project command line option, [252](#)
- force
 - loadpo command line option, [255](#)
- force-commit
 - pushgit command line option, [256](#)
- format {csv,json,text,html}
 - wlc command line option, [100](#)
- get-name
 - changesite command line option, [248](#)
- ignore
 - import_json command line option, [250](#)
- inconsistent
 - auto_translate command line option, [247](#)
- input
 - wlc command line option, [101](#)
- key KEY
 - wlc command line option, [100](#)
- lang LANGUAGE
 - loadpo command line option, [255](#)
- language-code
 - list_translators command line option, [254](#)
- language-map LANGMAP
 - import_memory command line option, [251](#)
- language-regex REGEX
 - import_project command line option, [252](#)
- license NAME
 - import_project command line option, [252](#)
- license-url URL
 - import_project command line option, [252](#)
- main-component
 - import_project command line option, [252](#)
- main-component COMPONENT
 - import_json command line option, [250](#)
- mt MT

auto_translate command line option, [247](#)
 -name
 createadmin command line option, [249](#)
 -name-template TEMPLATE
 import_project command line option, [252](#)
 -new-base-template TEMPLATE
 import_project command line option, [252](#)
 -no-password
 createadmin command line option, [249](#)
 -no-privs-update
 setupgroups command line option, [257](#)
 -no-projects-update
 setupgroups command line option, [257](#)
 -no-update
 setuplang command line option, [257](#)
 -optimize
 rebuild_index command line option, [256](#)
 -origin ORIGIN
 delete_memory command line option, [250](#)
 -output
 wlc command line option, [101](#)
 -overwrite
 auto_translate command line option, [247](#)
 wlc command line option, [101](#)
 -password PASSWORD
 createadmin command line option, [249](#)
 -project PROJECT
 import_json command line option, [250](#)
 -rebuild
 optimize_memory command line option, [256](#)
 -set-name NAME
 changesite command line option, [248](#)
 -source PROJECT/COMPONENT
 auto_translate command line option, [247](#)
 -threshold THRESHOLD
 auto_translate command line option, [247](#)
 -type {origin}
 list_memory command line option, [254](#)
 -update
 createadmin command line option, [249](#)
 import_json command line option, [250](#)
 install_addon command line option, [254](#)
 -url URL
 wlc command line option, [100](#)

-user USERNAME
 auto_translate command line option, [247](#)
 -username USERNAME
 createadmin command line option, [249](#)
 -vcs NAME
 import_project command line option, [252](#)
 .Net Resource
 file format, [67](#)

A

add_suggestions
 django-admin command, [247](#)
 add_suggestions command line option
 -author USER@EXAMPLE.COM, [247](#)
 ADMINS
 setting, [118](#)
 AKISMET_API_KEY
 setting, [216](#)
 ALLOWED_HOSTS
 setting, [118](#)
 Android
 file format, [63](#)
 ANONYMOUS_USER_NAME
 setting, [216](#)
 API, [76](#), [99](#), [103](#)
 Apple strings
 file format, [64](#)
 AUDITLOG_EXPIRY
 setting, [216](#)
 AUTH_LOCK_ATTEMPTS
 setting, [216](#)
 AUTH_TOKEN_VALID
 setting, [217](#)
 auto_translate
 django-admin command, [247](#)
 auto_translate command line option
 -add, [247](#)
 -inconsistent, [247](#)
 -mt MT, [247](#)
 -overwrite, [247](#)
 -source PROJECT/COMPONENT, [247](#)
 -threshold THRESHOLD, [247](#)
 -user USERNAME, [247](#)
 AUTO_UPDATE
 setting, [216](#)
 AUTOFIX_LIST
 setting, [218](#)
 AVATAR_URL_PREFIX
 setting, [217](#)

B

BASE_DIR

setting, 218
 bilingual
 translation, 59

C

celery_queues
 django-admin command, 248
 changes
 wlc command line option, 101
 changesite
 django-admin command, 248
 changesite command line option
 -get-name, 248
 -set-name NAME, 248
 CHECK_LIST
 setting, 218
 checkgit
 django-admin command, 248
 cleanup
 wlc command line option, 101
 cleanup_avatar_cache
 django-admin command, 249
 cleanuptrans
 django-admin command, 249
 Comma separated values
 file format, 67
 Command (*class in wlc.main*), 104
 COMMENT_CLEANUP_DAYS
 setting, 219
 commit
 wlc command line option, 100
 commit_pending
 django-admin command, 248
 commit_pending command line option
 -age HOURS, 248
 COMMIT_PENDING_HOURS
 setting, 219
 commitgit
 django-admin command, 248
 createadmin
 django-admin command, 249
 createadmin command line option
 -email USER@EXAMPLE.COM, 249
 -name, 249
 -no-password, 249
 -password PASSWORD, 249
 -update, 249
 -username USERNAME, 249
 CSV
 file format, 67

D

DATA_DIR
 setting, 219

DATABASES
 setting, 119
 DEBUG
 setting, 119
 DEFAULT_ACCESS_CONTROL
 setting, 220
 DEFAULT_COMMITER_EMAIL
 setting, 220
 DEFAULT_COMMITER_NAME
 setting, 220
 DEFAULT_FROM_EMAIL
 setting, 119
 DEFAULT_MERGE_STYLE
 setting, 221
 DEFAULT_PULL_MESSAGE
 setting, 221
 DEFAULT_TRANSLATION_PROPAGATION
 setting, 221
 delete_memory
 django-admin command, 250
 delete_memory command line option
 -all, 250
 -origin ORIGIN, 250
 django-admin command
 add_suggestions, 247
 auto_translate, 247
 celery_queues, 248
 changesite, 248
 checkgit, 248
 cleanup_avatar_cache, 249
 cleanuptrans, 249
 commit_pending, 248
 commitgit, 248
 createadmin, 249
 delete_memory, 250
 dump_memory, 250
 dumpuserdata, 250
 import_json, 250
 import_memory, 251
 import_project, 251
 importuserdata, 253
 importusers, 253
 install_addon, 253
 list_ignored_checks, 254
 list_languages, 254
 list_memory, 254
 list_translators, 254
 list_versions, 255
 loadpo, 255
 lock_translation, 255
 move_language, 255
 optimize_memory, 256
 pushgit, 256
 rebuild_index, 256

- setupgroups, 257
- setuplang, 257
- unlock_translation, 257
- updatechecks, 257
- updategit, 257
- download
 - wlc command line option, 101
- DTD
 - file format, 69
- dump_memory
 - django-admin command, 250
- dumpuserdata
 - django-admin command, 250

E

- ENABLE_AVATARS
 - setting, 221
- ENABLE_HOOKS
 - setting, 221
- ENABLE_HTTPS
 - setting, 221
- ENABLE_SHARING
 - setting, 222
- environment variable
 - POSTGRES_DATABASE, 140
 - POSTGRES_HOST, 141
 - POSTGRES_PASSWORD, 140
 - POSTGRES_PORT, 141
 - POSTGRES_SSL_MODE, 141
 - POSTGRES_USER, 140
 - REDIS_DB, 141
 - REDIS_HOST, 141
 - REDIS_PASSWORD, 141
 - REDIS_PORT, 141
 - ROLLBAR_ENVIRONMENT, 142
 - ROLLBAR_KEY, 142
 - SENTRY_DSN, 142
 - SENTRY_ENVIRONMENT, 142
 - SENTRY_PUBLIC_DSN, 142
 - WEBLATE_ADMIN_EMAIL, 136, 138
 - WEBLATE_ADMIN_NAME, 136
 - WEBLATE_ADMIN_PASSWORD, 135, 136
 - WEBLATE_AKISMET_API_KEY, 138
 - WEBLATE_ALLOWED_HOSTS, 137
 - WEBLATE_AUTH_LDAP_BIND_DN, 139
 - WEBLATE_AUTH_LDAP_BIND_PASSWORD, 139
 - WEBLATE_AUTH_LDAP_SERVER_URI, 139
 - WEBLATE_AUTH_LDAP_USER_ATTR_MAP, 139
 - WEBLATE_AUTH_LDAP_USER_DN_TEMPLATE, 139
 - WEBLATE_DEBUG, 136
 - WEBLATE_DEFAULT_FROM_EMAIL, 136
 - WEBLATE_EMAIL_HOST, 141
 - WEBLATE_EMAIL_HOST_PASSWORD, 141

- WEBLATE_EMAIL_HOST_USER, 141
- WEBLATE_EMAIL_PORT, 141
- WEBLATE_EMAIL_USE_SSL, 142
- WEBLATE_EMAIL_USE_TLS, 142
- WEBLATE_ENABLE_HTTPS, 137
- WEBLATE_GITHUB_USERNAME, 75, 138
- WEBLATE_GOOGLE_ANALYTICS_ID, 138
- WEBLATE_IP_PROXY_HEADER, 137
- WEBLATE_LOGIN_REQUIRED_URLS_EXCEPTIONS, 138
- WEBLATE_LOGLEVEL, 136
- WEBLATE_MT_DEEPL_KEY, 138
- WEBLATE_MT_GLOSBE_ENABLED, 138
- WEBLATE_MT_GOOGLE_KEY, 138
- WEBLATE_MT_MICROSOFT_COGNITIVE_KEY, 138
- WEBLATE_MT_MYMEMORY_ENABLED, 138
- WEBLATE_NO_EMAIL_AUTH, 140
- WEBLATE_REGISTRATION_OPEN, 137
- WEBLATE_REQUIRE_LOGIN, 138
- WEBLATE_SECRET_KEY, 137
- WEBLATE_SERVER_EMAIL, 136
- WEBLATE_SIMPLIFY_LANGUAGES, 138
- WEBLATE_SITE_TITLE, 136
- WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_KEY, 140
- WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_SECRET, 140
- WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY, 140
- WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET, 140
- WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID, 140
- WEBLATE_SOCIAL_AUTH_BITBUCKET_KEY, 139
- WEBLATE_SOCIAL_AUTH_BITBUCKET_SECRET, 139
- WEBLATE_SOCIAL_AUTH_FACEBOOK_KEY, 139
- WEBLATE_SOCIAL_AUTH_FACEBOOK_SECRET, 139
- WEBLATE_SOCIAL_AUTH_FEDORA, 140
- WEBLATE_SOCIAL_AUTH_GITHUB_KEY, 139
- WEBLATE_SOCIAL_AUTH_GITHUB_SECRET, 139
- WEBLATE_SOCIAL_AUTH_GITLAB_API_URL, 140
- WEBLATE_SOCIAL_AUTH_GITLAB_KEY, 140
- WEBLATE_SOCIAL_AUTH_GITLAB_SECRET, 140
- WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY, 139
- WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET, 139

139

WEBLATE_SOCIAL_AUTH_OPENSUSE, 140

WEBLATE_SOCIAL_AUTH_UBUNTU, 140

WEBLATE_TIME_ZONE, 137

WL_BRANCH, 213

WL_FILE_FORMAT, 213

WL_FILEMASK, 213

WL_LANGUAGE, 213

WL_NEW_BASE, 213

WL_PATH, 213

WL_PREVIOUS_HEAD, 213

WL_REPO, 212

WL_TEMPLATE, 213

WL_VCS, 212

F

file format

.Net Resource, 67

Android, 63

Apple strings, 64

Comma separated values, 67

CSV, 67

DTD, 69

Gettext, 60

il8next, 66

Java properties, 62

Joomla translations, 62

JSON, 65

PHP strings, 64

PO, 60

Qt, 63

RC, 69

RESX, 67

Ruby YAML, 68

Ruby YAML Ain't Markup Language, 68

string resources, 63

TS, 63

XLIFF, 61

YAML, 68

YAML Ain't Markup Language, 68

G

get() (*wlc.Weblate method*), 103

Gettext

file format, 60

GITHUB_USERNAME

setting, 222

GOOGLE_ANALYTICS_ID

setting, 222

H

HIDE_REPO_CREDENTIALS

setting, 222

I

il8next

file format, 66

import_json

django-admin command, 250

import_json command line option

-ignore, 250

-main-component COMPONENT, 250

-project PROJECT, 250

-update, 250

import_memory

django-admin command, 251

import_memory command line option

-language-map LANGMAP, 251

import_project

django-admin command, 251

import_project command line option

-base-file-template TEMPLATE, 252

-file-format FORMAT, 252

-language-regex REGEX, 252

-license NAME, 252

-license-url URL, 252

-main-component, 252

-name-template TEMPLATE, 252

-new-base-template TEMPLATE, 252

-vcs NAME, 252

importuserdata

django-admin command, 253

importusers

django-admin command, 253

importusers command line option

-check, 253

install_addon

django-admin command, 253

install_addon command line option

-addon ADDON, 253

-configuration CONFIG, 253

-update, 254

IP_BEHIND_REVERSE_PROXY

setting, 222

IP_PROXY_HEADER

setting, 222

IP_PROXY_OFFSET

setting, 223

iPad

translation, 64

iPhone

translation, 64

J

Java properties

file format, 62

Joomla translations

file format, 62

JSON
file format, [65](#)

L

LEGAL_URL
setting, [223](#)

LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH
setting, [223](#)

list-components
wlc command line option, [100](#)

list-languages
wlc command line option, [100](#)

list-projects
wlc command line option, [100](#)

list-translations
wlc command line option, [100](#)

list_ignored_checks
django-admin command, [254](#)

list_languages
django-admin command, [254](#)

list_memory
django-admin command, [254](#)

list_memory command line option
-type {origin}, [254](#)

list_translators
django-admin command, [254](#)

list_translators command line option
-language-code, [254](#)

list_versions
django-admin command, [255](#)

load() (*wlc.config.WeblateConfig method*), [104](#)

loadpo
django-admin command, [255](#)

loadpo command line option
-force, [255](#)
-lang LANGUAGE, [255](#)

lock
wlc command line option, [101](#)

lock-status
wlc command line option, [101](#)

lock_translation
django-admin command, [255](#)

LOGIN_REQUIRED_URLS
setting, [223](#)

LOGIN_REQUIRED_URLS_EXCEPTIONS
setting, [223](#)

ls
wlc command line option, [100](#)

M

MACHINE_TRANSLATION_SERVICES
setting, [224](#)

main() (*in module wlc.main*), [104](#)

monolingual

translation, [59](#)

move_language
django-admin command, [255](#)

MT_APERTIUM_APY
setting, [224](#)

MT_AWS_ACCESS_KEY_ID
setting, [224](#)

MT_AWS_REGION
setting, [225](#)

MT_AWS_SECRET_ACCESS_KEY
setting, [225](#)

MT_BAIDU_ID
setting, [225](#)

MT_BAIDU_SECRET
setting, [225](#)

MT_DEEPL_KEY
setting, [225](#)

MT_GOOGLE_KEY
setting, [225](#)

MT_MICROSOFT_COGNITIVE_KEY
setting, [225](#)

MT_MYMEMORY_EMAIL
setting, [226](#)

MT_MYMEMORY_KEY
setting, [226](#)

MT_MYMEMORY_USER
setting, [226](#)

MT_NETEASE_KEY
setting, [226](#)

MT_NETEASE_SECRET
setting, [226](#)

MT_SAP_BASE_URL
setting, [227](#)

MT_SAP_PASSWORD
setting, [227](#)

MT_SAP_SANDBOX_APIKEY
setting, [227](#)

MT_SAP_USE_MT
setting, [227](#)

MT_SAP_USERNAME
setting, [227](#)

MT_SERVICES
setting, [224](#)

MT_TMSERVER
setting, [226](#)

MT_YANDEX_KEY
setting, [226](#)

MT_YOUDAO_ID
setting, [227](#)

MT_YOUDAO_SECRET
setting, [227](#)

N

NEARBY_MESSAGES

setting, [228](#)

O

optimize_memory

django-admin command, [256](#)

optimize_memory command line option

-rebuild, [256](#)

P

PHP strings

file format, [64](#)

PIWIK_SITE_ID

setting, [228](#)

PIWIK_URL

setting, [228](#)

PO

file format, [60](#)

post() (*wlc.Weblate method*), [104](#)

pull

wlc command line option, [100](#)

push

wlc command line option, [100](#)

pushgit

django-admin command, [256](#)

pushgit command line option

-force-commit, [256](#)

Python, [103](#)

Q

Qt

file format, [63](#)

R

RATELIMIT_ATTEMPTS

setting, [217](#)

RATELIMIT_LOCKOUT

setting, [217](#)

RATELIMIT_WINDOW

setting, [217](#)

RC

file format, [69](#)

rebuild_index

django-admin command, [256](#)

rebuild_index command line option

-clean, [256](#)

-optimize, [256](#)

register_command() (*in module wlc.main*), [104](#)

REGISTRATION_CAPTCHA

setting, [228](#)

REGISTRATION_EMAIL_MATCH

setting, [228](#)

REGISTRATION_OPEN

setting, [228](#)

repo

wlc command line option, [101](#)

reset

wlc command line option, [100](#)

REST, [76](#)

RESX

file format, [67](#)

RFC

RFC 4646, [58](#)

Ruby YAML

file format, [68](#)

Ruby YAML Ain't Markup Language

file format, [68](#)

S

SECRET_KEY

setting, [119](#)

SERVER_EMAIL

setting, [119](#)

SESSION_ENGINE

setting, [119](#)

setting

ADMINS, [118](#)

AKISMET_API_KEY, [216](#)

ALLOWED_HOSTS, [118](#)

ANONYMOUS_USER_NAME, [216](#)

AUDITLOG_EXPIRY, [216](#)

AUTH_LOCK_ATTEMPTS, [216](#)

AUTH_TOKEN_VALID, [217](#)

AUTO_UPDATE, [216](#)

AUTOFIX_LIST, [218](#)

AVATAR_URL_PREFIX, [217](#)

BASE_DIR, [218](#)

CHECK_LIST, [218](#)

COMMENT_CLEANUP_DAYS, [219](#)

COMMIT_PENDING_HOURS, [219](#)

DATA_DIR, [219](#)

DATABASES, [119](#)

DEBUG, [119](#)

DEFAULT_ACCESS_CONTROL, [220](#)

DEFAULT_COMMITER_EMAIL, [220](#)

DEFAULT_COMMITER_NAME, [220](#)

DEFAULT_FROM_EMAIL, [119](#)

DEFAULT_MERGE_STYLE, [221](#)

DEFAULT_PULL_MESSAGE, [221](#)

DEFAULT_TRANSLATION_PROPAGATION, [221](#)

ENABLE_AVATARS, [221](#)

ENABLE_HOOKS, [221](#)

ENABLE_HTTPS, [221](#)

ENABLE_SHARING, [222](#)

GITHUB_USERNAME, [222](#)

GOOGLE_ANALYTICS_ID, [222](#)

HIDE_REPO_CREDENTIALS, [222](#)

IP_BEHIND_REVERSE_PROXY, [222](#)

IP_PROXY_HEADER, [222](#)

- IP_PROXY_OFFSET, 223
 - LEGAL_URL, 223
 - LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH, 223
 - LOGIN_REQUIRED_URLS, 223
 - LOGIN_REQUIRED_URLS_EXCEPTIONS, 223
 - MACHINE_TRANSLATION_SERVICES, 224
 - MT_APERTIUM_API, 224
 - MT_AWS_ACCESS_KEY_ID, 224
 - MT_AWS_REGION, 225
 - MT_AWS_SECRET_ACCESS_KEY, 225
 - MT_BAIDU_ID, 225
 - MT_BAIDU_SECRET, 225
 - MT_DEEPL_KEY, 225
 - MT_GOOGLE_KEY, 225
 - MT_MICROSOFT_COGNITIVE_KEY, 225
 - MT_MYMEMORY_EMAIL, 226
 - MT_MYMEMORY_KEY, 226
 - MT_MYMEMORY_USER, 226
 - MT_NETEASE_KEY, 226
 - MT_NETEASE_SECRET, 226
 - MT_SAP_BASE_URL, 227
 - MT_SAP_PASSWORD, 227
 - MT_SAP_SANDBOX_APIKEY, 227
 - MT_SAP_USE_MT, 227
 - MT_SAP_USERNAME, 227
 - MT_SERVICES, 224
 - MT_TMSERVER, 226
 - MT_YANDEX_KEY, 226
 - MT_YOUDAO_ID, 227
 - MT_YOUDAO_SECRET, 227
 - NEARBY_MESSAGES, 228
 - PIWIK_SITE_ID, 228
 - PIWIK_URL, 228
 - RATELIMIT_ATTEMPTS, 217
 - RATELIMIT_LOCKOUT, 217
 - RATELIMIT_WINDOW, 217
 - REGISTRATION_CAPTCHA, 228
 - REGISTRATION_EMAIL_MATCH, 228
 - REGISTRATION_OPEN, 228
 - SECRET_KEY, 119
 - SERVER_EMAIL, 119
 - SESSION_ENGINE, 119
 - SIMPLIFY_LANGUAGES, 229
 - SITE_TITLE, 229
 - SPECIAL_CHARS, 229
 - STATUS_URL, 229
 - SUGGESTION_CLEANUP_DAYS, 229
 - URL_PREFIX, 229
 - WEBLATE_ADDONS, 229
 - WEBLATE_FORMATS, 230
 - WEBLATE_GPG_IDENTITY, 230
 - setupgroups
 - django-admin command, 257
 - setupgroups command line option
 - no-privs-update, 257
 - no-projects-update, 257
 - setuptools
 - django-admin command, 257
 - setuptools command line option
 - no-update, 257
 - show
 - wlc command line option, 100
 - SIMPLIFY_LANGUAGES
 - setting, 229
 - SITE_TITLE
 - setting, 229
 - SPECIAL_CHARS
 - setting, 229
 - statistics
 - wlc command line option, 101
 - STATUS_URL
 - setting, 229
 - string resources
 - file format, 63
 - SUGGESTION_CLEANUP_DAYS
 - setting, 229
- ## T
- translation
 - bilingual, 59
 - iPad, 64
 - iPhone, 64
 - monolingual, 59
 - TS
 - file format, 63
- ## U
- unlock
 - wlc command line option, 101
 - unlock_translation
 - django-admin command, 257
 - updatechecks
 - django-admin command, 257
 - updategit
 - django-admin command, 257
 - upload
 - wlc command line option, 101
 - URL_PREFIX
 - setting, 229
- ## V
- version
 - wlc command line option, 100
- ## W
- Weblate (*class in wlc*), 103
 - WEBLATE_ADDONS

- setting, 229
- WEBLATE_ADMIN_EMAIL, 136, 138
- WEBLATE_ADMIN_NAME, 136
- WEBLATE_ADMIN_PASSWORD, 135, 136
- WEBLATE_EMAIL_USE_SSL, 142
- WEBLATE_EMAIL_USE_TLS, 142
- WEBLATE_FORMATS
 - setting, 230
- WEBLATE_GITHUB_USERNAME, 75
- WEBLATE_GPG_IDENTITY
 - setting, 230
- WeblateConfig (*class in wlc.config*), 104
- WeblateException, 103
- wlc, 99
- wlc (*module*), 103
- wlc command line option
 - config PATH, 100
 - config-section SECTION, 100
 - convert, 101
 - format {csv,json,text,html}, 100
 - input, 101
 - key KEY, 100
 - output, 101
 - overwrite, 101
 - url URL, 100
- changes, 101
- cleanup, 101
- commit, 100
- download, 101
- list-components, 100
- list-languages, 100
- list-projects, 100
- list-translations, 100
- lock, 101
- lock-status, 101
- ls, 100
- pull, 100
- push, 100
- repo, 101
- reset, 100
- show, 100
- statistics, 101
- unlock, 101
- upload, 101
- version, 100

wlc.config (*module*), 104

wlc.main (*module*), 104

X

XLIFF

- file format, 61

Y

YAML